Volume 11 Issue 03 Published, August 22, 2007 ISSN 1535-864X DOI: 10.1535/itj.1103

Intel® Technology Journal

Tera-scale Computing

The theme of the eight papers in this Q3'07 Intel Technology Journal is "Tera-scale Computing Research." Future tera-scale computers will be based on 10s to 100s of integrated processor cores. We discuss the research work in progress at Intel's labs including its architectural design, heterogeneous multi-threaded programming, on-package integrated memory, tera-scale runtime programming environment, data center on a chip and applications in media mining and physics graphics.

Inside you'll find the following articles:

More information, including current and past issues of Intel Technology Journal, can be found at:
http://developer.intel.com/technology/itj/index.htm

Volume 11 Issue 03 Published, August 22, 2007 ISSN 1535-864X DOI: 10.1535/itj.1103

Intel® Technology Journal

Tera-scale Computing

Articles

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

By Lin Chao
Editor and Publisher, *Intel Technology Journal*

Engineers and researchers in Intel's advanced research labs have been investigating ideas associated with tera-scale (10^{12}) computing power (trillions of operations per second) to help solve highly complex problems, do critical mathematical analysis, or run computationally intensive applications more efficiently and in real time. This investigation is based on the growing need for intensive computation, visualization, or manipulation and management of massive amounts of data. We anticipate trillions of calculations to occur within a second (teraflops) to achieve performance and productivity to run these complex scientific and commercial applications.

Our devices will be able to comprehend data better and use this knowledge to act on our behalf. To do so, computers must have the ability to think in terms of models–digital "models" of people, places, and information.

The theme of the eight papers in this Q3'07 *Intel Technology Journal* is "Tera-scale Computing Research." Future tera-scale computers will be based on 10s to 100s of integrated processor cores. We discuss the research work in progress at Intel's labs. The first six papers look at tera-scale architecture research and discuss both its inherent challenges and some potential solutions to those challenges. In particular, the first paper looks at tera-scale architecture and design tradeoffs. With its very high level of integration and the presence of heterogeneous building blocks, a modular and scalable **on-chip interconnect** is required. Based on the organization, architectural building blocks, and physical design constraints, we expect ring, 2D-mesh, or similar topologies to be an attractive option. The second paper reviews future tera-scale architecture's inclusion of accelerator cores alongside Intel® Architecture cores. We show how an **accelerator exoskeleton** can provide a shared Virtual Memory (VM) heterogeneous multi-threaded programming paradigm for these accelerators by adding extensions to the CPU instruction set and software tools that have an Intel Architecture (IA) look-n-feel. The third paper describes the package technology required for tera-scale computing needs. The scope and focus of the paper are primarily design and electrical performance challenges. We discuss a roadmap that evolves from today's off-package memory to **complex on-package integrated memory architectures**. The fourth paper presents the design and implementation of a runtime environment for tera-scale platforms. We discuss the design and implementation of a **Many-Core RunTime (McRT) environment**—a prototype tera-scale runtime environment. We present simulation results from a tera-scale simulator to show that McRT enables excellent scalability on tera-scale platforms. The fifth paper proposes a hardware scheme to accelerate **dynamic task scheduling**. To harness the computing resources of tera-scale computing architecture cores, applications must expose their thread-level parallelism to the hardware. We explore hardware schedulers that do this for large-scale multiprocessor systems: they decompose parallel sections of programs into many tasks and let a task scheduler dynamically assign tasks to threads. In the sixth paper we start by highlighting tera-scale potential in **datacenter environments**. We show how a multi-tier datacenter workload that required

tens (to hundreds) of platforms in the past can potentially map onto one (or a few) single-socket tera-scale platforms running VMs and thereby create Datacenter-on-Chip (DoC) architectures.

The remaining papers then review tera-scale applications on media mining and physical simulations. **Media mining** can help us more easily retrieve, organize, and manage the exponentially growing amount of media data. The seventh paper explores several usage models in media mining. To efficiently use the processing power provided by multi-core processors, we studied common parallelization schemes and propose a **general parallel framework** for these media-mining applications. And finally, in the eighth paper of this Q3'07 *Intel Technology Journal*, we study physical simulation applications in two broad categories: production physics and game physics. After parallelization, the benchmark applications achieve parallel scalabilities of thirty to sixty times on a simulated chip-multiprocessor with 64 cores.

So, how does tera-scale computing affect our lives at home? In our future homes, we will use tera-scale computers for better personal media management, immersive and personalized entertainment, educational collaborations and personal health visualization and management, to name a few examples. I am especially excited that this issue of the *Intel Technology Journal* has this topic as its focus: I am part of the team working on tera-scale computing research in Intel's labs. Tera-scale computing will play a major role in shaping Intel's computer architecture for the next decade. We are excited to continue our work to realize these possibilities.

Foreword

By:

Sean Koehl, Technology Strategist, Intel Corporate Technology Group

Jerry Bautista, Director of Technology Management, CTG Microprocessor Technology Laboratory

Jim Held, Intel Fellow, Corporate Technology Group, Director, Tera-Scale Computing Research, Intel Corporation

Ram Huggahalli, Principal Engineer/Network Platform Architect, CTG Communication Technology Lab

The coming of the microprocessor in 1968 began a new age. With this tool we have vastly accelerated our ability to learn, design, and innovate. With each new silicon process generation, faster transistors and microarchitectural innovations have enabled computational power to grow dramatically, opening new possibilities and opportunities and moving from microprocessors as simple calculators to multi-media and communication devices, that connect us to a world-wide web filled with information, entertainment and a new kind of electronic community. Software execution sped up almost automatically as clock speeds increased and the world learned to purchase based on MHz, then GHz.

However, we have now reached a turning point, where we must rely more on another benefit of each new silicon process, the doubling of transistor count each generation known as Moore's Law. To maintain the pace of increasing computational capability with the energy efficiency needed for mobility and within the power and thermal limits of large datacenters, we find ourselves as an industry turning from increased frequency to parallelism. The power efficiency inherent in dividing work among multiple processor cores on one die allows us to continue dramatic increases in performance. In the next decade or so, processors with not just two or four, but 10s or 100s of compute cores will be the norm with a commensurate increase in performance.

Aside from simply providing the same tera-scale computation at lower cost, these parallel microprocessors will surpass the massively parallel supercomputers of the past. Tera-bits of inter-processor I/O bandwidth, exceptionally low core-core latency, and a pool of integrated system resources will enable new applications that are hard to imagine today, just as the early PC user doing text-based word processing would be astounded by the digital content creation capabilities of today's machines.

We can see on the horizon a new class of model-based applications enabled by parallel computing. Computers will have the ability to form increasingly sophisticated digital "models" of things and ideas. This will give applications the ability to recognize, find, and even synthesize real or virtual people, places, and things amidst a sea of data. Computers' ability to process and analyze data will reach a point where they may no longer be considered simple tools: they will act more as our trusted advisors and assistants – anticipating needs without being "asked."

This is the motivation of Intel's Tera-scale Computing Research program. In order to enter this new and challenging era of innovation, we must create new architectures, based on 10s or even 100s of cores. We must develop not just new hardware, but new system and application software to effectively manage and execute more and more threads at the same time. Alongside academia and our industry fellow travelers, we must

challenge the previous way of doing things, while at the same time finding a smooth path to a new world of mainstream parallel computing. It is not overdramatic to say we are entering a renaissance in computing.

In this issue of the *Intel Technology Journal,* we share some of our most recent findings with you. The topics, ranging from hardware to software, represent a few key projects from the many underway at Intel. We hope that the results presented here not only assist in other research and development efforts, but that they inspire new projects and new ideas on parallel microprocessing. Our tera-scale computing vision is aggressive, with a potentially huge ROI; it will take thousands of minds to realize the future hundred-core chips with their billions of transistors.

Technical Reviewers for Q3 2007 ITJ

Allen Baum, Digital Enterprise Group
Henning Braunisch, Technology and Manufacturing Group
Bob S. Dreyer, Mobility Group
Pepe Gonzalez, Corporate Technology Group
Mohammad R. Haghighat, Software and Solutions Group
Yatin Hoskote, Corporate Technology Group
Bo Huang, Software and Solutions Group
Ram Huggahalli, Corporate Technology Group
Brian Lewis, Corporate Technology Group
Geoff Lowney, Software and Solutions Group
Ravi V. Mahajan, Technology and Manufacturing Group
Greg J. Regnier, Corporate Technology Group
Arch Robison, Software and Solutions Group
Kumar Shiv, Software and Solutions Group
Byoungro So, Corporate Technology Group
Hariharan Thantry, Corporate Technology Group
Xinmin Tian, Software and Solutions Group

Integration Challenges and Tradeoffs for Tera-scale Architectures

Mani Azimi, Corporate Technology Group, Intel Corporation
Naveen Cherukuri, Corporate Technology Group, Intel Corporation
D. N. Jayasimha, Corporate Technology Group, Intel Corporation
Akhilesh Kumar, Corporate Technology Group, Intel Corporation
Partha Kundu, Corporate Technology Group, Intel Corporation
Seungjoon Park, Corporate Technology Group, Intel Corporation
Ioannis Schoinas, Corporate Technology Group, Intel Corporation
Aniruddha S. Vaidya, Corporate Technology Group, Intel Corporation

Index words: tiled architecture, on-die interconnect, cache hierarchy, communication protocol

ABSTRACT

Tera-scale processors promise to offer an unprecedented concentration of computing power and enable novel usages and applications. The computing power may be provided by a combination of general-purpose cores and special-purpose (fixed or programmable) computing engines. Further, Moore's law enables the integration of additional system resources to the processor die. However, the realization of tera-scale architecture is challenged by on-die power dissipation, wire delays, off-chip memory bandwidth, process variations, and higher failure rates. These challenges create opportunities for architectural innovation. One of the ways to address these challenges is through the use of a "tiled" architecture: the die is divided into a large number of identical or close-to-identical, tiles that are interconnected using a scalable and energy-efficient interconnect. This modular approach enables ease of layout and rapid integration of different blocks. Limited off-chip memory bandwidth requires innovations in the cache hierarchy, memory subsystem, and coherence protocol. We present an architectural vision for the tera-scale processors and discuss the performance, scalability, and manufacturability aspects of the uncore. We articulate key challenges and point to candidate solutions for these challenges.

INTRODUCTION

Over the last few years, dual-core processors have become mainstream in desktop, mobile, and server platforms due to their ability to deliver higher system performance more efficiently than single-core processors. The trend towards higher core counts is continuing strong with quad-core processors establishing an increasing presence across all market segments.

Industry experience with small-scale shared memory multiprocessors enabled a relatively effortless integration of a small number of processors into a single die. Moving beyond a small number to tens or hundreds of processor cores at the same time as other platform ingredients such as memory controllers, I/O bridges, and graphics engines find their way to the processor die, introduces significant challenges to the infrastructure that ties all these together. This infrastructure includes the on-die interconnect, the cache hierarchy, the memory, the I/O, and system interfaces. In this paper we use the term *uncore* to collectively refer to all the elements in the processor die that are not computing engines.

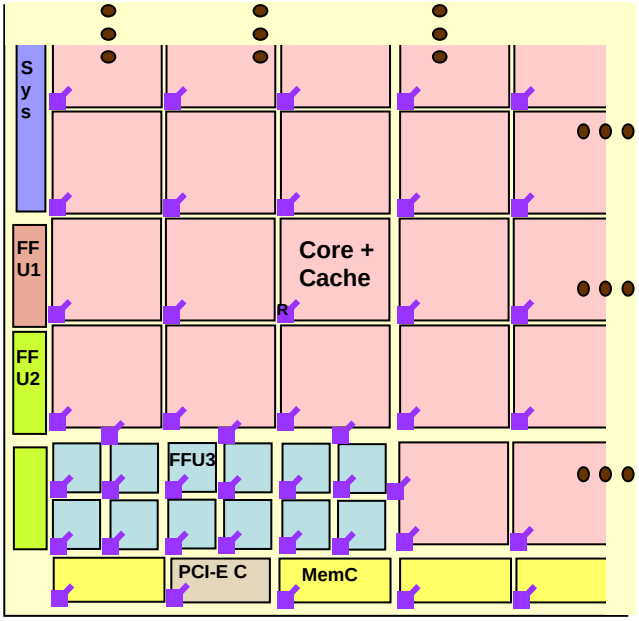
The tera-scale architecture uncore must be capable of satisfying the communication requirements of a large number of cores, fixed function computing engines, and the external memory and I/O system. In order to scale effectively, the uncore must find ways to keep the off-die bandwidth manageable and within the constraints of cost, power, and high-speed signaling technology. The uncore must be able to offer significant flexibility to assign computing resources to concurrently solve different problems. It must include mechanisms to enable high-volume manufacturing by enhancing reliability in the presence of increasing architectural complexity and

decreasing silicon geometries. Moreover, it must perform its functions within a constrained power envelope.

This paper is organized as follows. First, we describe the architectural vision for tera-scale processors. Second, we focus on the challenges and opportunities imposed by the tera-scale architecture in the key uncore elements such as the on-die interconnect, cache hierarchy, and memory architecture. We conclude with a summary of the key challenges, opportunities, and directions outlined in this paper.

ARCHITECTURAL VISION

The tera-scale architectural vision, as shown in Figure 1, takes the integration trend to its logical progression by consolidating not only a large number of general-purpose computing cores but also special-purpose computing engines (e.g., texture units, shader units, fixed function units), and platform elements, such as memory and I/O controllers, in a single die. A tera-scale processor may also include a system interface to allow multiple such processors to connect with each other and with other system peripherals.



FFU: Fixed Function Unit, **Mem C:** Memory Controller, **PCI-E C:** PCI-based Controller, **R:** Router, **Sys I/F:** System Interface

Figure 1: Tera-scale architecture: high-level block diagram

The tera-scale architecture uncore consists of the following key elements:

- A scalable high-bandwidth, low-latency, and power-efficient interconnect to connect the computing and platform elements together and allow them to exchange information with each other, access memory, and communicate with the rest of the system.
- A cache hierarchy that allows the multiple computing elements to effectively utilize and share the on-die memory resources.
- A scalable, high-bandwidth memory architecture that can effectively feed the large number of computing elements.

We expect tera-scale processors to be highly optimized for specific market segments through variations in the number of computing engines, by having different types of fixed function blocks, and having a different type and number of memory and I/O resources. Not all building blocks require the high bandwidth and low latency offered by the scalable interconnect. We expect blocks that are not candidates for integration into the main interconnect and cache hierarchy to be attached to auxiliary interconnects suitable for specific needs.

ON-DIE INTERCONNECT

The on-die interconnect is the primary "meeting ground" for various elements of the tiled architecture in Figure 1. Given its central nature, there are certain basic requirements for the on-die interconnect:

- **Scalability**: Given the requirements of a large number of nodes (agents) on the interconnect (high tens to low hundreds), we realistically desire a) a sub-linear growth in average distance with number of nodes, b) a relatively low per-hop latency through each switch under no-load conditions, and c) manageable growth in latency under loaded conditions.
- **Partitionability**: The topology, with appropriate routing support, should enable the tera-scale architecture to be dynamically partitioned to achieve both performance and fault isolation.
- **Fault tolerance**: The tera-scale architecture with its tiled structure has the potential for a graceful degradation under faults. Further, with the expected impact of variations on process technology, there is a greater susceptibility to "performance" faults (discussed in the next section). Hence, the topology, with appropriate support, should support routing around faults.
- **Validation and testing**: The interconnect should provide support for testing and validation, which is critical for high-volume manufacturing. For example, an interconnect that uses a deadlock-free routing approach is easier to test and validate compared to one using deadlock-recovery based routing.

- **Regularity**: In order to make the design of the tera-scale chip tractable, it is imperative that the layout, planning, and design of each tile be done in such a way as to make the tile physically symmetric. Thus integration may be achieved largely through abutment of tiles. To that end, each "tile" needs to plan its global wiring tracks.
- **Flexibility and design friendliness**:
  - Designs should facilitate "choppability" so that with minimal redesign effort, a range of market segments can be satisfied.
  - Furthermore, the basic router design should not change as the underlying parameter, for example, as the number of processors, changes with each process generation.
  - The tiled architecture will have different-sized tiles arising possibly from the need for heterogeneous cores (e.g., some suited for throughput and others suited for single-thread performance), specialized engines, fixed function units, etc. The on-die interconnect design needs to incorporate the needs of each by, for example, clustering multiple low-bandwidth engines into a single routing agent.

Candidate Topologies

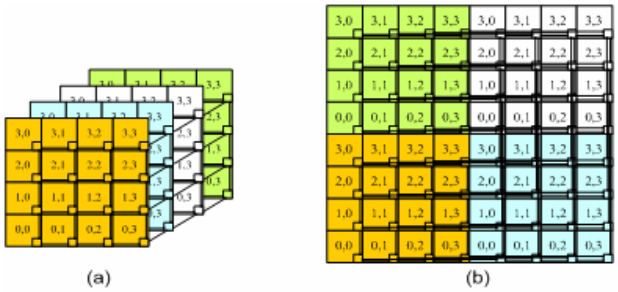


Figure 2: 2D embedding of a 64-node 3D-mesh network

In off-chip networks, both the number of links (which has a bearing on the topology) and the link widths are determined by pin-out limitations. In on-chip networks, this limitation is absent. The topology choices (apart from the number of routing agents that need to be supported) are determined by the wiring density and router complexity in terms of area and power. The wiring density is in turn determined by the number of metal layers available and the directionality constraints (uniform availability of horizontal and vertical metal layers), as well as the need for the topology to be embedded in 2D space. The latter point implies that for higher (greater than 2D) dimensional networks, *topological adjacency does not lead to spatial adjacency* [14]. This has significant implications both on the wire delay and on the wiring density. Consider the embedding of the 3D mesh in Figure 2. For the longest hop, the topological distance is 9, but three of these hops span half the length of the die. Hence, the distance in tile span units is 18!

Considering wiring density, router complexity, and design friendliness, tera-scale architecture topologies will be fixed-degree and will have a low dimension (1–2) in the foreseeable future. Thus, ring and 2D torus/mesh networks and their many variants [2] will be candidate topologies.

In the rest of this section, we use the 2D mesh as an example topology for illustrative purposes only.

Interconnect Microarchitecture

The main challenge in on-die networks is to achieve the required bandwidth and latency under the constraints of power and area. While topological choices, as mentioned above, help with bandwidth scaling and keep latencies manageable, they come at increasing power costs.

Wang et al. [28] show that the router power is almost 2x the power of the wires in the MIT-RAW [25] chip. Further decomposition shows that the power is roughly spent as much in the switch (crossbar) as in the buffers.

Wang et al. also propose segmented and cut-through crossbars as possible solutions to reduce crossbar power. Meanwhile Nicopoulos et al. [21] reduce buffer power through careful microarchitectural techniques to minimize the number of buffers required for the same network throughput.

Kumar et al. [16] observe that certain paths in a 2D mesh are common for a number of flows (between different source/destination pairs), and thus traffic traveling on these trunks could be aggregated and switched together—thus avoiding the need for packets to stop and be buffered at intermediate nodes. This in turn saves buffer power and reduces contention on network resources—the latter helping to improve the throughput and thus eventually the energy characteristics of the network.

Traffic Classes

Emerging workloads [10] may see different classes of traffic overlaid on the on-die network. Taylor et al. [25] and Gratz et al. [11] use a network fabric to route operands between different clusters of functional units. It is conceivable that the different cores of the tera-scale processor may be used to realize a virtual superscalar microarchitecture [29], thus necessitating fine-grained communication of operands and control, in addition to the cache coherent and message-based communication in the cache-memory subsystem.

Furthermore, as media applications become a dominant consumer of compute capacity on a chip, they will place hard real-time constraints on different shared resources such as cache and interconnect. In addition, running disparate applications on the same multi-core is likely to result in bandwidth over-subscription by some applications at the cost of starvation of some others. Careful rationing of bandwidth while providing latency guarantees to the necessary applications will require careful architecture definition and design of the fabric.

Resiliency

The need for fault tolerance arises from both an increased susceptibility to faults and the opportunity to gracefully degrade in a tera-scale environment.

Future process technology trends are likely to adversely affect the resilience of a tera-scale processor chip. Such trends might include process variations becoming a more significant determinant of overall performance and insufficient burn-in time to weed out infant mortality. Consequently, there is a higher probability of in-field failures and accelerated degradation potentially shortening the expected lifetime of the product [6].

Interconnect Fault-tolerance Approaches

The following mechanisms can be adopted for addressing resilience in a tera-scale processor interconnect. These approaches can be used either to address true faults or performance faults (i.e., when underperforming or "out-of-spec" tiles are treated as failed tiles).

Sparing: Spare processor tiles paired with network interfaces and switches can potentially solve a multiplicity of fault scenarios including increased possibility of in-field failures. Upon detection of failures in some tile components, spare tiles are activated after the interconnection network is reconfigured as shown in Figure 3.

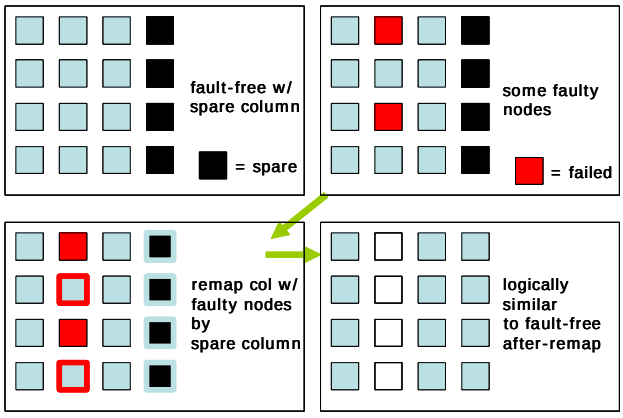


Figure 3: Illustrating use of spare tiles that maintain original topology

Fault-tolerant Routing

Fault-tolerant routing support is required in the interconnect to enable reconfiguration of the system components in the presences of failed tiles and routers. Upon system reset/initialization, a fault and topology discovery algorithm is run to determine the location/identity of failed components and to mark them in the interconnect. Other regions also need to be marked safe or unsafe from a deadlock-free routing perspective. A fault-tolerant routing algorithm is then configured to route around faulty and unsafe regions. Figure 4 shows faulty (dead) nodes in the interconnect. A few additional nodes are marked unsafe so as to form rectangular fault regions. After the fault-tolerant routing algorithm (such as in [5]) is configured, all working (and spare nodes, if sparing is used) tiles in the fabric can communicate with each other.

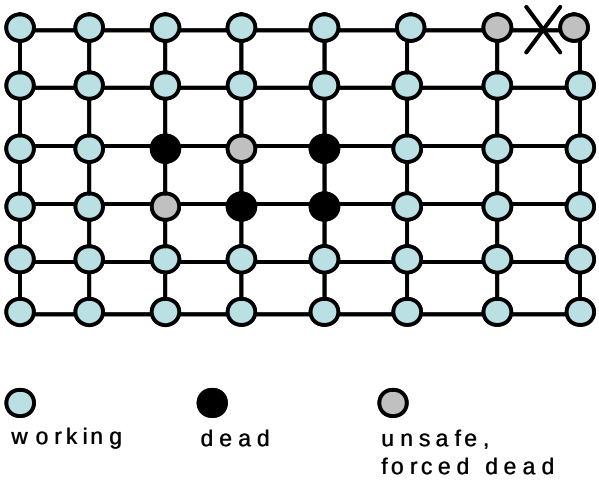


Figure 4: Illustrating need for fault-tolerant routing

The fault-tolerant routing algorithm should be simple to implement, deadlock free, and be able to handle a wide variety of faults. It is also desirable for the routing algorithm to adaptively respond to congestion that may occur in the network due to the additional effort needed to route around fault regions.

Partitioning for Performance Isolation

We expect several partitions to be supported on a tera-scale processor—each partition with a fraction of the total number of processing units, special-purpose units, and other platform elements. There may be several different usage models for a partitioned tera-scale processor including multiple server partitions in a consolidated "server on chip" or, for example, multiple virtual appliances on a home server.

It is desirable that the performance of each of the multiple partitions on a tera-scale processor be unaffected by the performance of other partitions. Some partitions may be more sensitive to performance perturbations from

other partitions or may have stricter Quality of Service (QoS) requirements.

Performance isolation: Performance isolation relies on confining intra-partition communication of a given QoS sensitive partition to physically distinct components of the on-die interconnect from that of other partitions. Figure 5 shows a configuration with three isolated partitions where traffic generated in one partition does not interfere with traffic from another partition.

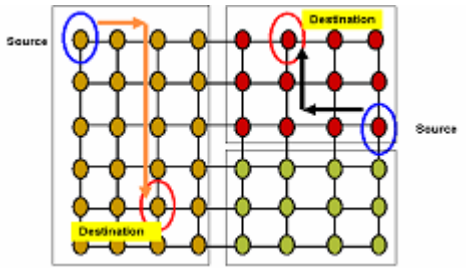


Figure 5: Performance isolation in a 2D mesh with rectangular partitions

Virtualization of Network Interfaces

In order to better realize a uniform interface that can comprehend the diverse needs of accelerators, fixed function units, general-purpose processor cores, cache blocks, etc., it is desirable to formalize and possibly even export the interconnect as an abstraction to the application programmer and/or the run-time system. Thus, one could envision a programmer fine-tuning an application's inter-processor communication requirements, based specifically on that application. For example, a media application requiring little support for cache coherence may be better served in power and performance through a direct send/receive interface. Similarly, the programmer may want to commandeer different levels of resources of the interconnects, i.e., number of buffers, switching priority, or bandwidth, and leave the rest of the hardware for use by another application.

A network interface that can allow this level of control and flexibility, yet can achieve good performance would be powerful. In addition, such a network makes for easy and rapid integration of multiple IP blocks that conform to the same interface.

CACHE HIERARCHY AND COHERENCE PROTOCOL

Diversity of workloads and concentration of compute resources in the tera-scale architecture put tremendous demands on the cache hierarchy and coherency protocol. This requires a flexible cache organization that can adapt to workload demands and puts minimal restrictions on the software to fully realize the performance potential. The associated coherency protocol needs to be efficient and scalable. It should also be flexible in terms of the requirements it imposes on the building blocks of the tera-scale architecture. In this section we highlight the challenges and tradeoffs associated with the cache hierarchy and protocol and point out potential directions for tera-scale architecture.

Developing parallel applications to harness and effectively use the massively parallel tera-scale processors is likely to be the key challenge for tera-scale computing. Many parallel programming models and languages have been deployed in different contexts over the last few decades and in fact, parallel programming remains an area of active research. A clear lesson, however, that we can draw from the history of parallel computing to date, is that hardware shared memory has proven to be a particularly successful programming model for general-purpose systems. Accordingly, tera-scale architecture should include first-class hardware support for shared memory. Industry and academic experience with coherence protocols for large-scale, hardware-shared memory machines has demonstrated that shared memory machines scaling to hundreds of processors can be successfully built. In fact, implementing a message-passing library such as the Message Passing Interface (MPI) over hardware-shared memory often results in higher bandwidth and lower latency than equivalent implementations using specialized low-latency cluster networks [19]. In addition, hardware support for shared memory will allow tera-scale processors to support common operating systems assuming that such operating systems overcome any existing scalability bottlenecks to harness the capabilities of tera-scale architecture.

A cache hierarchy should efficiently support a wide range of programming models and workloads. These are some important classes:

- Multiprogrammed workloads where there is no communication and data sharing among the processes running in different cores.
- Workloads with a mix of scalar and parallel sections. The performance of these workloads on tera-scale architecture is limited by the performance of the scalar section as indicated by Amdahl's law.
- Highly parallel workloads, where most of the computations can be parallelized. These workloads may exhibit one or more of the types of parralelism as described below:
    - Thread parallelism: Each thread may be similar or very different from each other and may or may not share data with other threads. Threads are created based on the granularities exposed by the application and then scheduled on available hardware contexts through task queues or other

constructs. Examples of this programming model can be found in transaction processing and Web applications.

- Data parallelism: A similar task is performed on different data sets, where some data may be shared between tasks. Applications are more structured, and algorithms are typically modified to fit the underlying cache organization. The number of threads used in this model is typically the same or less than the number of hardware contexts available. Examples of this programming model can be found in media, numerical analysis, and data-mining workloads.
- Streams: Programs are structured as kernels where input data are processed and output data are fed into other kernels. In this model threads (or kernels) are statically scheduled to hardware contexts. Within each kernel, thread- or data-level parallelism constructs can be applied to break tasks into ever finer sizes. Examples of this programming model can be found in media and graphics applications.

Cache Organization

A combination of different workloads and different types of parallelisms within these workloads presents unique architectural and design challenges for the cache hierarchy of tera-scale architecture. Architectural challenges center on the organization and policies associated with the cache hierarchy to meet performance, scalability, and energy-efficiency goals. Cache organization deals with the number of levels in the cache hierarchy, and with the size, associativity, latency, and bandwidth parameters at each level. Cache policies determine accessibility, allocation, and eviction policies to effectively utilize on-chip cache resources.

The objective of a cache hierarchy is to minimize the latency to frequently accessed data. In a traditional uniprocessor cache hierarchy, we move cache blocks closer and closer to the core through the levels in the cache hierarchy, based on access frequency. The same principle applies to multi-core cache hierarchies, but we have to take into account whether cores have to share a given level in the cache hierarchy or whether a level is implemented as a single physical block or as multiple physically distributed banks with non-uniform access latency to each bank.

In multi-core processors released over the last few years, the first one or two levels in the cache hierarchy are private to each core. However, different designs have pursued a range of options in sharing the last-level cache. In some designs such as those described in [20], the last-level cache is private to a core. In others, such as those described in [18, 23], the last-level cache is shared among multiple cores.

In CMPs with only a few cores, the last-level cache is being implemented as a single physical block with uniform access latency to the entirety of the cache by all the cores sharing it. As the number of cores and cache banks increase, physically distributed caches become attractive from a physical design perspective [15]. Moreover, by collocating a portion of the cache with a subset of the cores, there is an opportunity to reduce access latency to a portion of the cache, instead of offering equally high latency to all the cache. Figure 6 summarizes different multi-core cache organizations according to their suitability for the types of workloads, assuming a distributed multibank last-level cache.

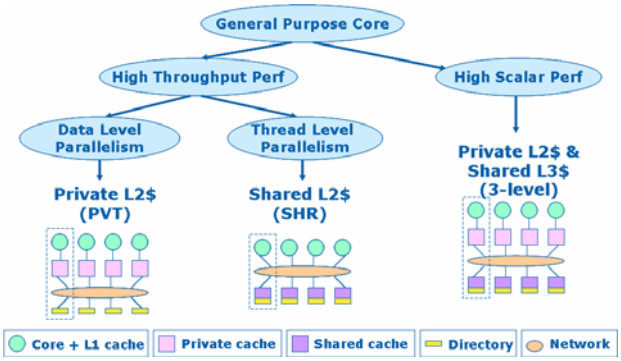


Figure 6: Cache organization options for multi-core architectures

In a tera-scale processor with a last-level cache physically distributed across multiple tiles, private and shared caches introduce distinct tradeoffs. A shared cache design increases effective cache capacity because only a single copy of a block shared by multiple cores resides in the cache. The downside is that any given block, whether private or shared, may be placed in a tile arbitrarily and be far away from the core(s) using it. In contrast, a private cache design will have all blocks used by a specific core on its local tile. However, since read-shared blocks will be replicated in multiple tiles, the effective cache capacity is reduced, and off-die traffic may increase.

Recent work suggests that other hybrid alternatives are possible: these combine the advantages of private and shared caches while avoiding their shortcomings. The key observation is that in a physically distributed cache design where some cache banks are closer to a specific core than others, one can optimize cache performance by optimizing the placement of blocks in the cache banks so that they are closer to the point of use. A number of approaches in the literature have been proposed to achieve this [4, 9, 30, 31]. Such approaches are beneficial in any multi-core processor with differential access

latency to a given portion of a shared cache, but are particularly effective in a tera-scale processor where there is large variation in the latency to access the cache in different tiles.

Fundamentally, all approaches have the following key policies to set: initial placement, read-shared block replication, block migration, and eviction. The initial placement policy defines where a block is placed in the cache hierarchy when it is fetched from memory. The replication policy determines whether multiple copies of a read-shared block can coexist in different cache banks. The block migration policy determines whether a block will move between tiles in response to processor accesses. Finally, the eviction policy determines what happens to a block evicted from a cache bank. Private and shared cache designs represent the end points in the design space with regard to these specific policies. For example, in a private design, a block is initially placed in the cache of the requesting core, while in a shared design, a block is placed in a cache bank determined by the physical address of the block (home tile). Hybrid approaches combine policies from private and shared design or introduce new policies to perform better than either private or shared designs, or they even dynamically switch between competing policies based on application demands. For example, the Adaptive Selective Replication (ASR) [4] determines the replication level within the context of a private cache design based on program behavior.

The enormous computing power available in tera-scale design implies that many applications (or applications consisting of many concurrent functions with distinct caching behavior) will be running concurrently (e.g., games physics with game AI and graphics rendering). Accordingly, when a level in the cache hierarchy is shared among multiple cores in the presence of diverse per-core access patterns and working sets, destructive interference can occur. One of the causes of destructive interference is the suboptimal behavior of the least recently used (LRU) replacement policy, typically implemented in processor caches, when the application workload exceeds the cache capacity. Sharing a cache level among multiple threads can further exacerbate the problem. This is a well known issue for any shared cache, including page disk and file system caches. Recent work in this area, however, shows some promise of success [22].

In its generalized version, the tera-scale architecture is a collection of modular and heterogeneous building blocks with well defined interfaces. Such a heterogeneous collection of elements puts its own unique requirements on the cache hierarchy, and meeting these with a single set of caching policies and a single cache hierarchy is quite challenging. For example, if an incarnation of tera-scale architecture is a collection of several general-purpose processors, some graphics coprocessors, a few network accelerators, a security coprocessor and so on, each of these processors exhibit very different data footprints and locality characteristics. Satisfying their needs through a unified cache hierarchy is challenging and requires further exploration.

Cache Coherency

The cache coherency protocol for tera-scale architecture must be scalable to a large number of caching agents and must enable efficient utilization of on-chip resources. The choice of a coherency protocol is closely linked to the cache organization and the interconnect. For example, a protocol designed for cache organization without any shared caches may be designed to keep precise information about the lines present in private caches, such that off-chip reads and writes are minimal. A snoop broadcast protocol is suitable when there is a broadcast interconnect, but it cannot be scaled.

On-chip interconnects are capable of providing an order of magnitude smaller latency and an order of magnitude higher bandwidth than off-chip socket-to-socket interconnects. Therefore, latency and bandwidth optimizations may not seem to be the primary goals for an on-chip coherency protocol. However, since tera-scale processors are expected to have a concentrated density of computing throughput, they do impose tremendously high bandwidth demands on the interconnect. Since the power delivery, cooling, and off-chip bandwidth available to each chip is not scaling with process technology, the protocol must enable improved utilization of on-chip cache structures, and the interconnect overhead, because of the protocol, must be kept to a minimum to gain the maximum performance under these limits.

Directory-based protocols have been widely used in large-scale, multichip multiprocessors [7, 8, 17, 24], where a directory is used to keep track of copies of blocks in different caches. The same concept can be applied to on-chip cache coherence protocols in tera-scale architecture with some modification. As illustrated in Figure 7, a directory consists of entries corresponding to lines in caches where each entry has a state field and a field to store the identities (indicated as pointers in the directory structure in Figure 7) of the caches with a copy of the block. The state field indicates if a block may be present in one of the caches and the possible states the cached copies could be in. For a directory that is inclusive of all the on-chip caches, a directory miss or a state of I (invalid) in the directory indicates that none of the caches have a copy of the block; a state of S (shared) indicates that some caches may have copies of the block in Shared state; and a state of X (exclusive) indicates that one of the caches may have a copy in either Modified, Exclusive, or

Shared state. When a directory entry is in S or X state, the identity field identifies the cache(s) with a copy of the memory block. The identity information can be stored in various ways, either as a set of bits with 1 bit for each cache (called full bit map), 1 bit for a group of caches (called coarse bit map) or a limited set of explicit cache identities with a mechanism to handle overflows. The cost of a full bit map directory may be acceptable for the first few generations of tera-scale architecture; however, a more compact representation may be desirable for further scalability.

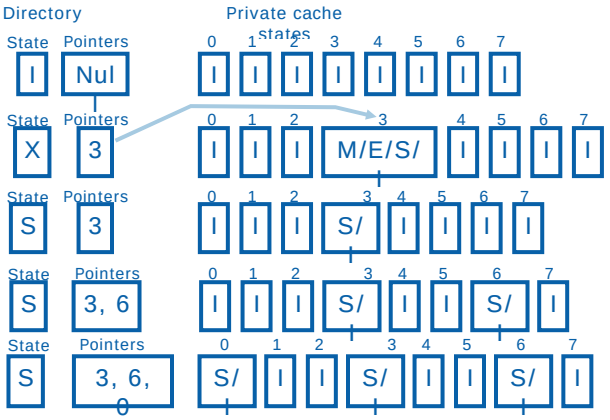


Figure 7: Directory structure to track cache lines

Since the purpose of the directory is to keep track of copies of a cache line in different private caches, the size, associativity, and replacement policy of the directory needs to provide adequate coverage for the total capacity of the private caches. Some designs combine the directory information in the same structure as the cache at the higher level in the hierarchy (if there is one), which may reduce complexity and area at the expense of some performance disadvantage due to conflicting policy requirements on the directory and cache. The cost and scalability or directory structures may start becoming a problem when the number of entities being tracked becomes very large. At that point, mechanisms to reduce directory size [8] or distributed directory [12] implementations may have to be considered.

The enormous amount of computing resources in a tera-scale platform enable a richer set of interactions between the computer and the end user than previously was possible. These include speech, motion and gesture recognition, enhanced visual effects, etc. often within virtual worlds where multiple users directly interact with each other. Interactions with the physical world introduce real-time considerations, and the tera-scale architecture must properly address them. Caches, however, interact in unpredictable ways with real-time applications. For this reason, processors targeted to interactive applications often include hardware mechanisms to allow applications to control the caching behavior to the point where one can reason about their expected performance [1]. Accordingly, the tera-scale cache hierarchy should include support in the form of locking primitives or similar mechanisms to allow applications to keep critical data in the caches. The exact form of such support is an area of active research.

Tera-scale architecture may also require much tighter integration of off-chip memory and I/O interfaces to take full advantage of its compute capabilities. Therefore, the on-chip protocol must enable optimizations for efficiently accessing local memory and for interacting with other auxiliary engines such as special-purpose co-processors and I/O controllers.

FEEDING THE BEAST: MEMORY ARCHITECTURE

With substantial increases in the computation power on a single die, one faces the challenge of feeding it with enough data bandwidth. For a small class of applications where the memory footprint is small, the memory accesses will mainly be exercising the on-die caches. For the majority of applications, a major increase in off-chip memory bandwidth is required. This manifests itself in two ways: (1) providing power-efficient high-speed off-die I/O; (2) providing power-efficient high bandwidth DRAM access. The former has seen steady progress in the past decade, but not at the required pace. The latter may require a new look at DRAM core and I/O design.

The first step to addressing the memory bandwidth challenge can be more efficient storage or improved management of the on-die storage. For example, embedded DRAM [3] helps to increase the density of on-die storage compared to SRAM. Efficient management of on-die storage by avoiding duplication of data in the cache hierarchy, as discussed in the previous section, is another step in increasing the effective capacity of on-die storage.

Integration of DRAM, e.g., GDDR memory, inside the processor package can offer more control over the I/O channel and thus allow a higher bandwidth, compared to crossing of package to motherboard-connector-DIMM path. Recent works have demonstrated methods of using 3D stacked SRAM to offer a low capacity high-bandwidth option, e.g., Intel's tera-scale prototype [13] and IBM's work on 3-D integrated circuits [26]. The freedom in the footprint design of such SRAM devices enables power-efficient solutions; however, limited capacity of such devices limit their application. 3D stacking of multiple DRAM dies can improve the memory capacity, but requires dense Through Silicon Vias (TSVs) to allow the required concurrency of accesses to independent DRAM banks.

Tera-scale Architecture Prototype

The Intel® Teraflop processor [27] is a prototype of some of the elements of tera-scale architecture. The Teraflop processor realizes an 80-core prototype with a 2D-mesh interconnect architecture that reaches more than 1Tflops of performance dissipating less than 100W of power. This illustrates the potential of the tera-scale architecture and validates the efficacy of some of the architectural building blocks.

SUMMARY

Tera-scale architecture presents tremendous challenges and opportunities to take advantage of Moore's Law. As discussed in this paper, the architectural and design tradeoffs for tera-scale architecture are unique to this architecture. A very high level of integration and the presence of heterogeneous building blocks necessitate a modular and scalable on-chip interconnect. Based on the organization, architectural building blocks, and physical design constraints, we expect ring, 2D-mesh, or similar topologies to be an attractive option. Interconnects with switches, such as 2D-mesh, though better in utilizing wiring tracks, bring their own challenges in terms of achieving aggressive latency targets within an acceptable power budget.

With shrinking device geometries and resulting increases in process variability and device failure rates, careful consideration needs to be given to get maximum performance without excessive cost through overly conservative designs. A flexible on-chip interconnect can play a role in dealing with variability and in-field failures by adapting to an optimal operating configuration through provisioning for fault-tolerant routing. A flexible interconnect can also be used to provide additional functionality, such as improved quality of service and performance isolation, to make tera-scale architectures more useful.

Providing adequate memory and I/O bandwidth to satisfy the needs of large numbers of compute engines in tera-scale architecture is a major challenge. Some of these can be addressed through using on-chip caches more effectively such that the needs of off-chip memory bandwidth are reduced. A higher integration of system components on tera-scale architecture also reduces pressure on memory bandwidth by avoiding the need for I/O controllers and compute engines to exchange data through the caches rather than memory. Technological approaches to improve the available memory bandwidth are also an active area of exploration, ranging from 3D stacked memory to higher speed memory interfaces, but they do have their own challenges, such as limited memory capacity and higher power consumptions, respectively.

In conclusion, tera-scale architecture is definitely in the not-too-distant future of mainstream computer architecture. Its realization, however, poses some challenges and a rich set of problems for researchers both in academia and industry. Problems and some solution strategies related to the "uncore" have been presented in this paper.

ACKNOWLEDGMENTS

We thank Yatin Hoskote, Dennis Brzezinski, David James, Mani Ayyar, Ching-Tsun Chou, Rama Menon, Saikat (Roy) Saharoy, Theodore Tabe, Hari Thantry, and Jianping (Jane) Xu for their contributions to material presented in this paper.

REFERENCES

[1] J. Andrews and N. Baker, "XBOX 360 System Architecture," *IEEE Micro*, March–April 2006.

[2] J. Balfour and W. J. Dally, "Design Tradeoffs for Tiled CMP On-Chip Networks," *International Conference on Supercomputing*, June 2006.

[3] J. Barth et al., "A 500MHz Random Cycle 1.5ns-Latency, SOI Embedded DRAM Macro Featuring a 3T Micro Sense Amplifier," *IEEE International Solid-State Circuits Conference*, Feb. 2007.

[4] B. M. Beckman, M. R. Marty and D. A. Wood, "ASR: Adaptive Selective Replication for CMP Caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (Micro), Orlando, FL, December 2006.

[5] R.V. Bopanna and S. Chalasani, "Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks," *IEEE Trans. Computers*, vol. 44, no. 7, pp. 848–864, July 1995.

[6] S. Borkar, "Challenges in Reliable System Design in the Presence of Transistor Variability and Degradation," *IEEE Micro*, vol. 25, n. 6, pp. 10–16 Nov.–Dec. 2005.

[7] F. Briggs et. al., "Intel 870: A Building Block for Cost-Effective Scalable Servers," *IEEE Micro*, March–April 2002, pp. 36–47.

[8] D. Chaiken, C. Fields, K. Kurihara, A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," *IEEE Computer*, June 1990, pp. 49–58.

[9] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," in *Proceedings of the 33rd International Symposium on Computer Architecture*, Boston, MA, June 2006.

[10] "Compute-Intensive, Highly Parallel Applications and Uses," *Intel Technology Journal*, Volume 09 Issue 02, May 2005.

[11] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S. Keckler, D. Burger, "Implementation and Evaluation of a Dynamically Routed Processor Operand Network," *IEEE/ACM International Symposium on Networks-on-Chips* (NOCS), May 2007.

[12] "IEEE standard for Scalable Coherent Interface (SCI)," *IEEE P1596*, August 1993.

[13] Intel News Release, "Intel Develops Tera-Scale Research Chips," Sept 26, 2006, at http://www.intel.com/pressroom/archive/releases/20060926corp_b.htm.

[14] D. N. Jayasimha, B. Zafar, Y. Hoskote, "On-die Interconnection Networks: Why They are Different and How to Compare Them," *Technical Report*, Microprocessor Technology Lab, Corporate Technology Group, Intel Corp.

[15] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[16] A. Kumar, L-S. Peh, P. Kundu, N. Jha, "Express Virtual Channels: Towards the Ideal Interconnection Fabric," in *Proceedings 34th Annual International Symposium on Computer Architecture* (ISCA'07), pp. 150–161, June 2007.

[17] D. Lenoski, J. Laudon, T. Joe, D Nakahira, L Stevens, A. Gupta, and J. Hennessy, "The DASH Prototype: Implementation and Performance," In *Proceedings19th International Symposium on Computer Architecture*, pp. 92–103, Gold Coast, Australia, May 1992.

[18] A. S. Leon, et al., "A power-efficient high-throughput 32-thread SPARC processor," *IEEE International Solid-State Circuits Conference*, Feb. 2006.

[19] "MPI Performance Measurements" at http://www.llnl.gov/computing/mpi/mpi_benchmarks.html.*

[20] C. McNairy and R. Bhatia, "Montecito: A dual-core, dual-threaded Itanium® processor," *IEEE Micro*, March–April, 2005.

[21] C.A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, C.R. Das, "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers," *International Symposium On Microarchitecture* (MICRO'06) pp. 333–346, Dec. 2006.

[22] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely Jr., J. Emer, "Adaptive Insertion Policies for High Performance Caching," *International Symposium on Computer Arachitecture*, June 2007.

[23] N. Sakran, et al., "The Implementation of the 65nm Dual-Core 64b Merom Processor," *IEEE International Solid-State Circuits Conference*, Feb. 2007.

[24] P. Stenstrom, "A survey of cache coherence schemes for multiprocessors," *IEEE Computer*, pp. 12–24, June 1990.

[25] M. B. Taylor, W. Lee, S. Amarasinghe, A. Agarwal, "Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures," *International Symposium on High Performance Computer Architecture*, February 2003.

[26] A. W. Topol et al., "Three-dimensional integrated circuits," IBM Journal of Research and Development, vol. 50, no. 4/5, 2006, pp. 491–506.

[27] S. Vangal et al., "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," *IEEE International Solid-State Circuits Conference*, Feb. 2007.

[28] H-S. Wang, L-S. Peh, N. Jha, "Power-driven design of router microarchitectures in on-chip networks," *International Symposium On Microarchitecture* (MICRO'03), pp. 105–116, Nov. 2003.

[29] P. Wu, A. E. Eichenberger, A. Wang, P. Zhao, "An integrated simdization framework using virtual vectors," *International Conference on Supercomputing*, pp. 169–178, June 2005.

[30] M. Zhang and K. Asanovic, "Victim Migration: Dynamically Adapting Between Private and Shared CMP Caches," *MIT CSAIL Technical Report*, MIT-CSAIL-TR-2005-064, Cambridge, MA, October 2005.

[31] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," in *Proceedings 32nd International Symposium on Computer Architecture*, Madison, WI, June 2005.

AUTHORS' BIOGRAPHIES

Mani Azimi is a Senior Principal Engineer and Director of the Platform Architecture Research team in the Microprocessor Technology Laboratory in Intel's Corporate Technology Group. He received his Ph.D. degree from Purdue University. He joined Intel in 1990 and has worked on a wide range of platform architecture topics including system protocol, processor interface, MP cache controller architecture, and performance modeling/analysis. He is currently focusing on tera-scale computer architecture challenges. His e-mail is mani.azimi at intel.com.

Naveen Cherukuri is a Research Scientist within Intel's Microprocessor Technology Labs (MTL) in Santa Clara, California. At Intel, he has worked on the Itanium® processor design and performance analysis. His current research focuses on the cache organization for tera-scale architecture. He has an MSEE degree from the University of Arizona, Tucson. His e-mail is naveen.cherukuri at intel.com.

D. N. Jayasimha is a Principal Engineer in the Corporate Technology Group at Intel Corporation with research interests in multiprocessor architectures, interconnects, and performance analysis. Prior to joining Intel he was a faculty member in Computer Science at the Ohio State University. He received his Ph.D. degree from the University of Illinois at Urbana Champaign. His e-mail is jay.jayasimha at intel.com

Akhilesh Kumar is a Principal Engineer in Intel's Corporate Technology Group and leads the definition of protocols for on-chip and off-chip system interconnects. His research interests include cache organization, on-chip and off-chip interconnects, and interface protocols. He received his Ph.D. degree in Computer Science from Texas A&M University. His e-mail is akhilesh.kumar at intel.com.

Partha Kundu is a Senior Staff Research Scientist within Intel's Microprocessor Technology Labs (MTL) in Santa Clara, California. He was an architect of the Intel® Itanium® architecture and a Principal Architect on a DEC/Alpha microprocessor. His research interests include on-chip networks, memory system design, transactional memory, and performance simulation. He holds an M.S. degree from the State University of New York, Stony Brook. His e-mail is partha.kundu at intel.com.

Seungjoon Park is a Research Scientist within Intel's Microprocessor Technology Labs (MTL) in Santa Clara, California. At Intel, he has contributed to the definition and formal verification of off-die and on-die cache coherence and system interface protocols. Prior to Intel, he worked at NASA Ames Research Center with the High-Assurance Software Design Research team on Java PathFinder, a system to verify executable Java bytecode programs. He received his Ph.D. degree in Electrical Engineering with a Minor in Computer Science from Stanford University, where he investigated the cache coherence protocol of the Stanford FLASH multiprocessor and developed operational memory models of SPARC V9 architecture. His e-mail is seungjoon.park at intel.com.

Ioannis (Yannis) Schoinas is a Principal Engineer in Intel's Corporate Technology Group. He received his B.S. and M.S. degrees from the University of Crete-Heraclion and his Ph.D. degree from the University of Wisconsin-Madison. At Intel he has worked on a wide range of platform architecture topics including coherence protocol, memory RAS, system partitioning, configuration management, system security, and virtualization. He is currently focusing on tera-scale computer architecture challenges. His e-mail is ioannis.t.schoinas at intel.com.

Aniruddha S. Vaidya. is a Research Scientist at Intel's Microprocessor Technology Labs (MTL) in Santa Clara, California. His contributions at Intel include workload characterization, performance analysis, and architecture of server platforms. His current focus is on router and interconnection network architecture for Intel's tera-scale computing initiative. Ani has B.Tech and M.Sc. (Engg.) degrees from Banaras Hindu University and the Indian Institute of Science, and a Ph.D. degree in Computer Science and Engineering from the Pennsylvania State University. His e-mail is aniruddha.vaidya at intel.com.

Accelerator Exoskeleton

Perry Wang, Corporate Technology Group, Intel Corporation
Jamison Collins, Corporate Technology Group, Intel Corporation
Gautham Chinya, Corporate Technology Group, Intel Corporation
Hong Jiang, Mobility Group, Intel Corporation
Xinmin Tian, Software Solutions Group, Intel Corporation
Milind Girkar, Software Solutions Group, Intel Corporation
Lisa Pearce, Mobility Group, Intel Corporation
Guei-Yuan Lueh, Mobility Group, Intel Corporation
Sergey Yakoushkin, Corporate Technology Group, Intel Corporation
Hong Wang, Corporate Technology Group, Intel Corporation

ABSTRACT

To maximize performance and power efficiency, future multi-core architectures may be heterogeneous, incorporating some accelerator cores alongside the IA cores. Accelerator Exoskeletons provide a shared virtual memory heterogeneous multi-threaded programming paradigm for these accelerators using novel CPU instruction set extensions and software tool chains with an Intel® Architecture (IA) look-n-feel. Firstly, we introduce the proposed architectural extensions known as the *Exoskeleton Sequencer* (EXO), which represents heterogeneous accelerators as ISA-based MIMD architecture resources, and a shared virtual memory heterogeneous multi-threaded program execution model that tightly couples specialized accelerator cores with general-purpose CPU cores. Then we introduce the *C for Heterogeneous Integration* (CHI) programming environment that includes a compiler, runtime, debugger, and performance-analysis tools. The CHI compiler extends the OpenMP pragma for heterogeneous multi-threading programming, and it produces a single fat binary with code sections corresponding to different instruction sets. The runtime can judiciously spread parallel computation across the heterogeneous cores to optimize performance and power.

INTRODUCTION

The relentless pace of Moore's Law will lead to mainstream multi-core microprocessor designs with extensive on-die integration of a large number of cores [11]. Fundamentally, to scale multi-core processor designs to incorporate a large number of cores, ultra low Energy Per Instruction (EPI) cores are essential [6]. One approach to improving EPI by an order of magnitude is through heterogeneous multi-core design, in which some cores vary in functionality, instruction set (ISA), performance, power, and energy efficiency [14]. The key challenge then becomes how to accomplish such heterogeneous integration and achieve high performance while still maintaining the look-n-feel of the classic mainstream IA-based programming models and software ecosystem.

In this paper we present an overview of EXOCHI: *Exoskeleton Sequencer* (EXO), an architecture proposal to represent heterogeneous accelerators as ISA-based MIMD architectural resources, and *C for Heterogeneous Integration* (CHI), a programming environment that supports tightly coupled integration of heterogeneous cores. The EXO architecture supports the familiar POSIX shared virtual memory multi-threaded programming model for heterogeneous cores. Architecturally, the heterogeneous cores are exposed to the programmer as a new form of sequencer resource. They can be regarded as application-level MIMD functional units on which user-level threads, or *shreds*, encoded in the accelerator-specific ISA can execute. Having a shared virtual address space between the IA sequencer and accelerator sequencers facilitates code and data sharing and harmonizes cooperation between the concurrent shreds of different ISAs. Such a program is said to be *multi-shredded*.

The CHI integrated programming environment allows an application developer to inline blocks of accelerator-specific assembly or domain-specific language with traditional C/C++ code. The CHI compiler produces a single fat binary consisting of executable code sections corresponding to the different ISAs. CHI further extends the OpenMP pragmas [21, 23, 26] to allow the programmer to express thread-level parallelism by demarcating parallel regions of code targeting

heterogeneous accelerators. The CHI extensions to OpenMP support both fork-join and producer-consumer parallelism among the accelerator shreds and between the IA shreds and the accelerator shreds. The CHI runtime can judiciously spread the shreds across the heterogeneous sequencers dynamically to maximize throughput performance while minimizing power.

The rest of the paper is organized as follows. We first briefly review related work. We then introduce the EXO architecture that supports a shared virtual memory heterogeneous multi-threaded programming model. We then present an overview of the CHI integrated programming environment that extends the Intel® C++ Compiler, runtime, and tool chains to provide the familiar IA look-n-feel to program heterogeneous cores. To prototype the EXO architecture, we describe potential heterogeneous multi-core processors which combine an Intel® Core™2 Duo processor [27] and two possible accelerators: an 8-core 32-thread Intel® Graphics Media Accelerator (GMA) X3000 [10] or the Datastream Processing Engine (DPE) from a research Scalable Communication Core (SCC) prototype [8]. We demonstrate code examples and evaluate performance.

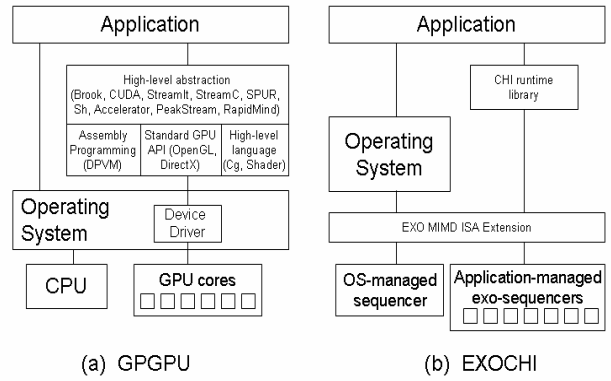


Figure 1: Alternate programming environments

RELATED WORK

There has been a rich body of research on heterogeneous acceleration. In most published work, the execution models usually fall into two classifications: (category 1) an ISA-based tightly coupled approach or (category 2), a device driver-based loosely coupled execution model. An example of the tightly coupled approach is the Software-configurable Processor (SCP) architecture [4] in which a custom ISA extension represents the operations implemented by a hardware accelerator attached to the CPU. The CPU is then responsible for sequencing, decoding, and dispatching each co-processor instruction, stalling until the co-processor execution completes. This approach resembles the classic x87 *escape-wait* style co-processor instruction execution where the co-processor does not sequence instructions independently from the CPU.

Examples of the second category include most known GPGPU infrastructures [1, 3, 5, 13, 15, 16, 17, 18, 19, 20, 22, 24, 25, 28]. As depicted in Figure 1(a), the CPU resources (cores and memory) are managed by the operating system (OS), and the GPU resources are separately managed by vender-supplied device drivers. Applications and device drivers run in separate address spaces, and consequently, data communication and synchronization between them is usually carried out in coarse granularity through explicit data copying via device driver APIs. In the EXOCHI framework depicted in Figure 1(b), the EXO architecture supports an execution model with a shared virtual address space and a POSIX multi-threaded programming model for the OS-managed IA sequencer and application-managed non-IA accelerator sequencers.

EXO differs from the existing tightly coupled approaches (category 1) by allowing independent sequencing and concurrent execution of multiple instruction streams on multiple sequencers within a single OS thread context. EXO also differs from the loosely coupled, driver-based approaches (category 2) by directly exposing the heterogeneous sequencers to application programs and by supporting a shared virtual address space amongst these sequencers. EXOCHI's user-level runtime can be used to schedule shreds and coordinate light-weight inter-shred data communication efficiently through shared virtual memory.

In addition, by supporting the shared virtual memory heterogeneous multi-threaded execution model, the CHI integrated programming environment enables the application developer to inline blocks of accelerator specific assembly or domain-specific languages within traditional C/C++ code. This allows performance sensitive parts of an algorithm to be optimized for the accelerator ISA just as Intel's SSE ISA extensions are traditionally used in implementing a high-performance math library. CHI's extensions to OpenMP allow programmers to express the underlying thread-level parallelism in a familiar parallel programming environment.

EXO ARCHITECTURE

Architecturally, EXO extends the Multiple Instruction Stream Processor (MISP) architecture [7] in three significant ways: (1) MISP exoskeleton (2) Address Translation Remapping (ATR), and (3) Collaborative Exception Handling (CEH). With this architectural support, EXO fundamentally enables a powerful shared virtual memory heterogeneous multi-threaded

programming model, despite ISA differences between the IA sequencer and the exo-sequencers.

MISP Exoskeleton

EXO provides a minimal architectural "wrapper," or exoskeleton, to make a non-IA heterogeneous accelerator sequencer conform to the MISP inter-sequencer signaling mechanism. With this exoskeleton, the accelerator sequencer can be exposed as an application-managed sequencer, even though it has a different ISA from the IA sequencers. To distinguish from an application-managed IA sequencer, we call such heterogeneous accelerator sequencers *exo-sequencers*. The exoskeleton supports interaction with the OS-managed IA sequencer through either initiating or responding to inter-sequencer user-level interrupts. With this enhancement, the code on an OS-managed IA sequencer can use MISP's **SIGNAL** instruction to dispatch shreds of a non-IA ISA to run on the exo-sequencers. This demands no additional OS support beyond MISP's requirements.

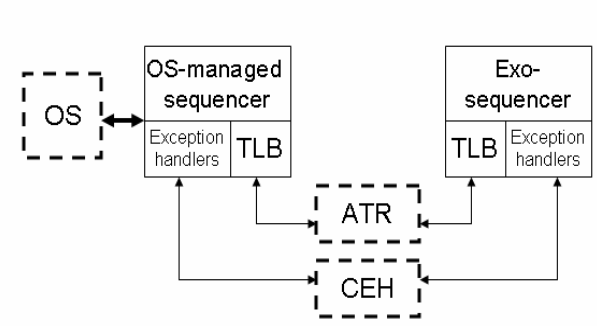


Figure 2: ATR and CEH between heterogeneous sequencers

Microarchitecture Support

Address Translation Remapping

To support shared virtual memory between the OS-managed IA sequencer and the exo-sequencers, EXO provides an ATR mechanism to allow the IA sequencer to handle page faults on behalf of the exo-sequencers.

Maintaining a shared virtual address space between two sequencers requires the same virtual address to be resolved to the same physical memory address on both sequencers. Among sequencers of the same architecture, this is accomplished by having the sequencers utilize the same page table for address translation. In a heterogeneous multi-core with IA sequencers and non-IA exo-sequencers, however, the page table format understood by each sequencer may differ. Directly accessing the IA page table is not an option for the exo-sequencers in such a case.

EXO solves this problem with its ATR mechanism. With ATR, when an exo-sequencer incurs a translation miss, it suspends shred execution and signals the IA sequencer to request *proxy execution* in order to service that Translation Lookaside Buffer (TLB) miss or page fault. Like MISP, upon receiving the proxy request as a user-level interrupt, the IA shred transfers control to a proxy handler that will touch the virtual address on behalf of the exo-sequencer. Once the page fault is serviced on the IA sequencer, however, unlike MISP, ATR will transcode the IA page table entry to the format of the exo-sequencer's page table entry before inserting the entry into the exo-sequencer's TLB. The exo-sequencer's TLB then points to the same physical page as the IA's TLB and can directly access the needed data. The exo-sequencer then resumes execution. As shown in Figure 2, an address translation remapping mechanism is responsible for remapping the IA page entry to the native format on the accelerator.

The shared virtual memory space for heterogeneous sequencers provides many benefits over the alternative approaches. It provides the essential architectural foundation to extend the classic shared memory multithreaded programming paradigm to heterogeneous multi-core processors. With a shared virtual address space, shreds from a single memory image executable running on IA sequencers and exo-sequencers can perform data communication and synchronization in familiar and efficient ways, e.g., without having to resort to explicit data copying as is necessary in the loosely-coupled approach.

It is important to note that even though ATR provides the necessary architectural support for a shared virtual address space, ATR by itself does not guarantee or require cache coherence between the IA sequencer and an exo-sequencer. In the absence of hardware support for cache coherence between the IA sequencer and an exo-sequencer, it is the responsibility of the programmer to use critical sections to protect other IA shreds from reading or writing the data being processed by shreds on the exo-sequencers. When an IA shred hands off a shared data structure to a shred on an exo-sequencer to process, the IA shred must first commit any dirty lines to main memory. Similarly, when the exo-sequencer shred completes its computation, it also needs to flush its cache before releasing a semaphore to the IA sequencer.

Clearly, with full cache coherence support between the IA sequencer and the exo-sequencer the programmer's work can be greatly eased. In particular, there is no need to use critical sections to ensure mutual exclusion on reads to the shared working set. This enables more concurrency between shreds on the IA sequencer and the exo-sequencer.

Collaborative Exception Handling

As with page faults, execution on the exo-sequencers could potentially incur exceptions or faults that require OS services. In conventional MISP, if an exception occurs on an application-managed sequencer, the instruction causing the exception can be replayed on the OS-managed sequencer through proxy execution. However, when the exception occurs on a non-IA exo-sequencer, the faulting instruction cannot simply be replayed on the IA CPU sequencer. Because the exo-sequencer uses a different ISA, the faulting instruction might have a data type that is not supported by IA ISA directly, or the exo-sequencer may require a different exception handling convention. To address this, EXO adds hardware support for CEH and a software-based exception handling mechanism, which allows faults or exceptions that occur on the exo-sequencer to be handled by the OS by proxy on the OS-managed IA sequencer.

Through CEH, an exception is handled in a similar fashion to a TLB miss. For example, as shown in Figure 2, when a double precision floating point vector instruction on an exo-sequencer incurs an exception, the exo-sequencer first signals the IA sequencer, as it does with ATR. The IA sequencer then functions as the proxy for the exo-sequencer by invoking an application-level handler to emulate the faulting vector instruction or use an OS service such as Structured Exception Handling (SEH) to provide full IEEE-compliant handling of the exception on the particular excepting scalar element. Once the exception is handled on the IA sequencer, CEH ensures the result is updated on the exo-sequencer before resuming execution.

Accelerator Exo-Sequencer: Two Examples

Media Accelerator

One example of an exo-sequencer accelerator is the integrated Intel Graphics Media Accelerator X3000 from the Intel® 965G Express chipset [9]. Figure 3 shows a high-level view of the GMA X3000 hardware. The GMA X3000 contains eight programmable, general-purpose graphics media accelerator cores, called Execution Units (EU), each of which supports four hardware thread contexts. From the programmer's perspective, 32 exo-sequencers are available. We use a custom emulation firmware that uses an IA CPU core as the OS-managed sequencer and uses the 32 GMA X3000 sequencers as exo-sequencers. The firmware implements all essential architectural extensions required by the EXO architecture, including MISP exoskeleton, ATR, and CEH.

A shred for the GMA X3000 exo-sequencer can be created either by an IA shred or spawned from another GMA X3000 shred. Once created, GMA X3000 shreds are scheduled in a software work queue in shared virtual memory like POSIX threads. The work queue can have a far greater number of shreds than the number of GMA X3000 exo-sequencers. The emulation firmware is responsible for translating a shred descriptor, which includes shred continuation information like instruction and data pointers to the shared memory, into implementation-specific hardware commands that the GMA X3000 exo-sequencers can consume and execute. The emulation layer hides all device-specific hardware details from the programmer.

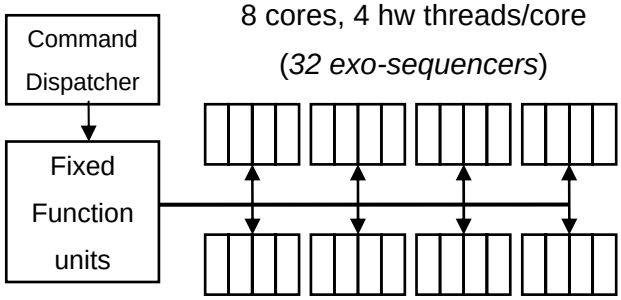


Figure 3: High-level view of the Intel GMA X3000

Communication Accelerator

Another example of the exo-sequencer accelerator is the Scalable Communication Cores (SCC) [8]. SCC is a research prototype designed for a reconfigurable radio baseband that is capable of processing several wireless standard protocols, such as WiFi, WiMax [12], or cellular infrastructure, with a common set of hardware. The SCC system architecture consists of a heterogeneous set of coarse-grained, highly optimized baseband Processing Elements (PEs).

One type of PE is the Data Processing Element (DPE) core, which performs computationally intensive operations, such as the Fast Fourier Transform (FFT) that is commonly used in many standard protocols. The DPE core structure consists of control and computation units and several memory blocks. DPE cores are connected via flexible interconnect matrices. Asynchronous data-path swap units support commutations from any of four inputs to any of four outputs. Reconfiguration of the data-path can be done dynamically with interconnection information and operation parameters stored in the configuration cache.

Inside DPE, there is a configuration (CFG) queue that is part of a special task scheduling mechanism. Each task pointer that is pushed onto the CFG queue will be fetched by the core engine. Each launched task becomes an exo-sequencer running on DPE. The DPE can be configured to use multiple CFG queues, thus implying a multi-threaded implementation. This allows multiple exo-sequencers to run concurrently on the DPE engine.

CHI PROGRAMMING ENVIRONMENT

C for Heterogeneous Integration (CHI) is designed to provide an IA look-n-feel programming environment to support user-level multi-shredding on heterogeneous sequencers. In the CHI infrastructure, we enhance the Intel C++ Compiler to support accelerator-specific inline assembly within the C/C++ source. In addition, we extend OpenMP pragmas to support heterogeneous multi-shredding and provide the related runtime support. The runtime library is responsible for judiciously scheduling heterogeneous shreds across the exo-sequencers. The compiler can also embed debugging information for different ISAs in a single binary. Such information can be used by an enhanced version of the Intel Debugger (IDB) to enable source-level debugging for both C/C++ code on the IA CPU target and the accelerator-specific code on the accelerator target. Figure 4 depicts the overall CHI compilation infrastructure. Three new capabilities are provided in the CHI compiler to allow programmers to express multi-shredded computation for the heterogeneous exo-sequencers in the C/C++ source code:

- A method to specify a region of accelerator-specific computation in either inline assembly or domain-specific language.
- A method to specify fork-join or producer-consumer style shred-level parallel execution for the inline accelerator-specific code region with OpenMP pragmas.
- A method to specify input and output memory regions and live-in values for the accelerator-specific code region.

Inline Accelerator Assembly Support

C/C++ provides a facility to inline assembly code blocks directly within the high-level source code. This capability provides programmers access to new instructions or processor features not exposed through the compiler and allows the most performance-critical parts of a program to be custom optimized in assembly. This inline assembly construct can be naturally extended to provide accelerator-specific inline assembly support.

Many variants of `asm` keyword and syntax exist. In CHI we adopt the Microsoft MASM syntax, i.e.,

`__asm {asm_statements;}`

where brackets are used to enclose the assembly statements. `__asm` is the keyword that indicates the enclosed block of code is a special assembly block written specifically for the given accelerator ISA. The *asm_statements* enclosed in the ensuing brackets are compiled into an accelerator-specific executable binary. The target ISA for the *asm_statements* is specified through the enclosing OpenMP pragma with the `target` clause, which is described in this paper in the section entitled "OpenMP Parallel Pragma Extension." As shown in Figure 4, a separate accelerator-specific assembler is dynamically linked with the Intel compiler. Figure 5 shows an example of C code using the extended OpenMP pragmas and CHI runtime APIs for a heterogeneous target consisting of an IA32 sequencer and GMA X3000 exo-sequencers.

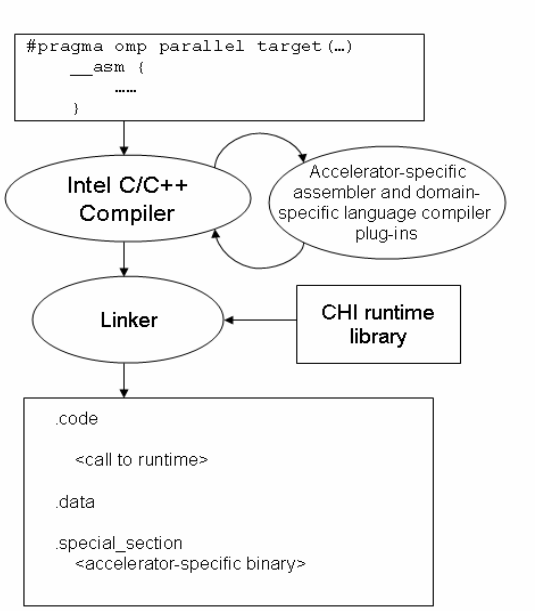


Figure 4: CHI compilation flow

Similar to traditional inline assembly, this accelerator-specific assembler generates code for the target ISA by translating the inline assembly instructions enclosed in the brackets into binary code and resolving symbolic names for memory locations and other entities referenced within the assembly block. After the assembler compiles the assembly block, the resulting binary code is embedded in a special code section of the executable indexed with a unique identifier. The final executable is a fat binary, consisting of binary code sections corresponding to different ISAs.

Domain-specific Language Support

In addition to supporting accelerator-specific inline assembly, the capability of the C/C++ compiler can be further extended to provide a facility to inline domain-specific language blocks directly within the high-level source code. These domain-specific languages are designed to utilize the accelerator-specific features not exposed through the general C/C++ programming environment. Therefore, the programmers can take advantage of the full capability of the underlying

accelerators without programming the exo-sequencer directly in assembly language.

```
int *A = malloc(n);
int *B = malloc(n);
int *C = malloc(n);

A_desc = chi_alloc_surface(A, X3000_INPUT, n, 1);
B_desc = chi_alloc_surface(B, X3000_INPUT, n, 1);
C_desc = chi_alloc_surface(C, X3000_OUTPUT, n, 1);
#pragma omp parallel target(x3000) shared(A,B,C)
    descriptor(A_desc,B_desc,C_desc) private(i)
{
  for (i=0; i<n/8; i++)
     __asm
     {
        shl.1.w    vr1 = i, 3
        ld.8.dw    [vr2..vr9] = (A, vr1, 0)
        ld.8.dw    [vr10..vr17] = (B, vr1, 0)
        add.8.dw   [vr18..r25] = [vr2..vr9], [vr10..vr17]
        st.8.dw    (C, vr1, 0) = [vr18..vr25]
     }
}
#pragma omp parallel for shared(D,E,F) private(i)
{
   for (i=0; i<n; i++)
      F[i] = D[i] + E[i];
}
```

Figure 5: Example GMA X3000 inline assembly

To provide a uniform programming interface to programmers, we adopt the format similar to that of the `asm` syntax, i.e.,

`__<language keyword>` {domain-specific language statements;}

where brackets are used to enclose the domain-specific language statements. `__<language keyword>` can be any language that is supported by CHI. Upon parsing the particular language keyword, the C/C++ compiler invokes the corresponding domain-specific compiler plug-ins to generate the accelerator-specific binary, similar to how it is done with the inline assembly support as described in the section entitled "Inline Accelerator Assembly Support."

Figure 6 shows an example of the domain-specific language support to the Data-stream Programming Language (DPL) that is specifically designed for the retargetable SCC-DPE accelerator. DPL provides essential high-level functions to exploit the inner microarchitecture of the DPE systolic arrays. The programmers can embed DPL code within the brackets preceded by the `__dpl` keyword.

OpenMP Parallel Pragma Extension

CHI extends the OpenMP `parallel` pragma. The construct for generating heterogeneous shreds of an accelerator-specific instruction set is outlined in Figure 7(a). The `target` clause specifies the particular accelerator instruction set used within the parallel region. The compiler inserts appropriate calls to the CHI runtime layer to enable judicious dynamic shred scheduling and dispatching onto the targeted exo-sequencers. When the main IA shred encounters an accelerator-specific `parallel` construct with the `target(`*`targetISA`*`)` clause, the IA shred spawns a team of `num_threads` heterogeneous shreds for the parallel region, where each shred eventually executes the enclosed assembly block on an exo-sequencer.

```
float Vin[4];
float Vout[4];

void *in_desc  = (void *)chi_alloc_buffer_desc
         (DPE_INPUT_BUFFER,  Vin, 4, 1);
void *out_desc = (void *)chi_alloc_buffer_desc
         (DPE_OUTPUT_BUFFER, Vout, 4, 1);

#pragma omp parallel target(dpe)
       shared(Vin,Vout)
descriptor(in_desc,out_desc)
{
  __dpl {
    configuration[1] cfgMult( vector val[1],
                    vector coeff[1] )
    {
      result bs( mull(val, coeff), 13 );
    }
    flow[4] multiFlow( vector vec[4],
           vector coeffs[4])
    {
      vector ret[4]; result out;
      selector[iter : 4] sel[1] = {{ iter }};
      selector[iter : 4] selRev[1] = {{ 3 - iter
}};
      ret[sel] = cfgMult(vec[sel], coeffs[selRev]);
    }
    vector cf[4] = { 0.5 + I * 0.0 };
    program dlMain()
    {
      Vout = multiFlow(Vin, cf);
    }
  }
}
```

Figure 6: Example inline DPL code using CH

By default, the main IA shred waits at the end of the construct until it is notified by the CHI runtime of the completion of all heterogeneous shreds. Similar to the traditional `nowait` clause, an optional `master_nowait` clause allows the main IA shred to continue execution past the construct after spawning the team of heterogeneous shreds, without having to wait for their completion. This allows concurrent execution on both the IA sequencer and its exo-sequencers. The CHI runtime is responsible for asynchronously notifying the IA sequencer of the eventual completion of all heterogeneous shreds.

OpenMP Work-Queuing Extension

In order to support concurrent threads with intricate dynamic inter-thread dependencies (e.g., due to the use of irregular data structures), the Intel C++ Compiler supports irregular parallelism through two special OpenMP pragmas, `taskq` and `task` [23]. In CHI, we further enhance the compiler and runtime to support inter-shred dependencies among heterogeneous shreds using these pragmas. The `parallel taskq` construct and the `task` construct for an exo-sequencer are outlined in Figure 7(b) and Figure 7(c).

```
#pragma omp parallel target(targetISA) [clause[[,]clause]…]
structured-block
```

Where clause can be any of the following:
firstprivate(*variable-list*)
private(*variable-list*)
shared(*variable-ptr-list*)
descriptor(*descriptor-ptr-list*)
num_threads(*integer-expression*)
master_nowait

(a) Parallel specification in fork-join threading model

```
#pragma intel omp taskq target(targetISA) [clause[[,]clause]…]
structured-block
```

Where clause can be any of the following:
firstprivate(*variable-list*)
private(*variable-list*)
shared(*variable-ptr-list*)
descriptor(*descriptor-ptr-list*)
num_threads(*integer-expression*)
master_nowait

(b) Queue specification in producer-consumer threading model

```
#pragma intel omp task target(targetISA) [clause[[,]clause]…]
structured-block
```

Where clause can be any of the following:
captureprivate(*variable-list*)
shared(*variable-ptr-list*)
descriptor(*descriptor-ptr-list*)

(c) Task specification in producer-consumer threading model

Figure 7: CHI extensions to OpenMP pragmas

CHI Runtime Support

The CHI runtime is a software library that translates the programmer-specified OpenMP directives into primitives to create and manage shreds that can carry out parallel execution on the heterogeneous multi-core target. Like conventional OpenMP runtimes, the CHI runtime layer provides a layer of abstraction that hides the details of managing the exo-sequencers from the programmer.

In order to allow the accelerator more efficient access to the C/C++ variables specified by the `shared` data clause, programmers can use the CHI runtime APIs to convey accelerator-specific access information through data structures known as descriptors. Descriptors are used by the accelerator to interpret the attributes of the `shared` variables that are accessed by the shreds.

EXOCHI PROTOTYPE

The EXOCHI framework described in this paper has already been deployed within Intel for successful development of production-quality, GMA X3000 media-processing kernels and other workloads of growing importance [2]. Figures 8 and 9 provide examples of the use of how an IA look-n-feel allows familiar development tools and environments to be used in writing heterogeneous multi-shredded code. Figure 8 shows the use of familiar legacy development tools (Microsoft Visual Studio*) for development and debugging of heterogeneous multi-shredded code. Figure 9 illustrates the compilation and execution of such a program.

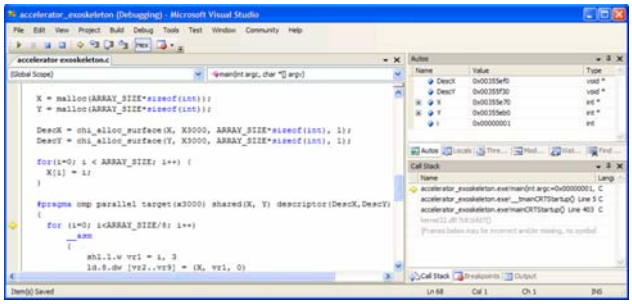


Figure 8: IA Look-n-Feel IDE (Microsoft Visual Studio) for application development

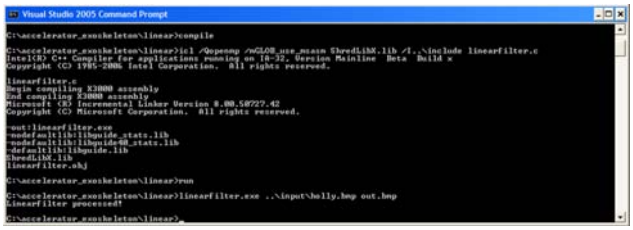


Figure 9: IA Look-n-Feel compilation and execution

Performance Evaluation

To evaluate the performance of our EXOCHI prototype we select a representative subset of the kernels that have been developed. These kernels exhibit a significant amount of data- and thread-level parallelism and thus, readily lend themselves to efficient execution on the GMA X3000 exo-sequencers.

Implementation of these kernels is made easy due to special GMA X3000 ISA features optimized for media processing. The key ISA features include wide SIMD instructions, predication support, and a large register file of 64 to 128 vector registers for each GMA X3000 exo-sequencer. With CHI, programmers can directly use the GMA X3000 ISA features via inline assembly in C/C++ code as if they are traditional ISA extensions to IA, such as SSE. By providing such IA look-n-feel, CHI enables highly productive development of heterogeneous multi-shredded code.

All benchmarks are compiled with the enhanced version of the Intel C++ Compiler using the most aggressive optimization settings (–fast –Qprof_use). These compiler optimizations include auto-vectorization, profile-guided optimization, and tune specifically for the Intel Core 2 Duo processor used in the EXO prototype system. `LinearFilter`, `SepiaTone` and `FGT` make use of the optimized and SSE-enhanced Intel IPP library, and the other benchmarks were manually tuned and SSE-optimized. Performance results measure the wall clock execution time.

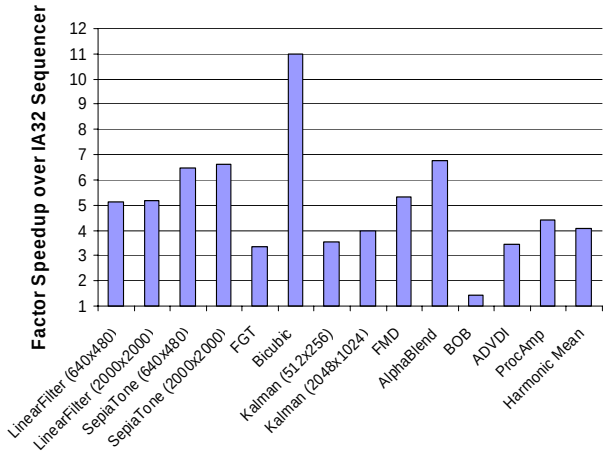


Figure 10: Speedup from execution on GMA X3000 exo-sequencers over IA sequence

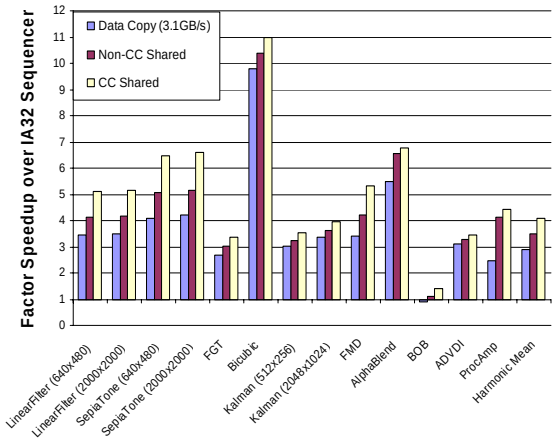


Figure 11: Impact of shared virtual memory

Performance Speedup on GMA X3000 Exo-sequencers over IA Sequencer

Figure 10 shows the speedup achieved over IA sequencer execution by executing media kernels on the GMA X3000 exo-sequencers. Significant speedup is achieved, ranging from 1.41X for `BOB` up to 10.97X for `Bicubic`. Two factors are crucial in achieving this high throughput performance on the GMA X3000 exo-sequencers. Most important is the availability of abundant shred-level parallelism. As each GMA X3000 exo-sequencer supports only in-order execution within a shred, the accelerator relies on the presence of multiple concurrent shreds to cover up stalls incurred in one shred by switching to another shred. A second, but related issue, is the need to maximize cache hit rate and memory bandwidth utilization. The GMA X3000 supports simultaneous execution of 32 hardware threads, each of which might be reading and writing multiple data streams. The CHI runtime allows programmers to carefully orchestrate shred scheduling to ensure shreds accessing adjacent or overlapping macroblocks are ordered closely together in the work queue so as to take advantage of spatial and temporal localities.

Other than support for thread-level parallelism, the GMA X3000 ISA also provides strong support for data-level parallelism. It features significantly wider SIMD operations (8- to 16-wide vector) than the SSE on today's IA CPU.

Impact of Data Copying Versus Shared Virtual Address Space

In general, the performance improvement achieved by using an accelerator is determined not only by the accelerator architecture but also by the overhead of data communication between the CPU and accelerator. This overhead varies greatly depending on the memory model between the CPU and the accelerator. Figure 11 shows overall performance improvement achieved with a cache coherent shared virtual memory model between the IA sequencer and the GMA X3000 exo-sequencers. In the absence of cache coherence or shared memory, the data communication overhead can significantly degrade the speedup achieved from accelerating the computation. In Figure 11 we contrast performance impacts for three memory model configurations.

The first configuration, *Data Copy*, assumes a model without shared virtual memory and no cache coherence between the IA sequencer and the GMA X3000 exo-sequencers. Consequently, data communication between IA shred and GMA X3000 shreds requires explicit data copying, for which we assume a 3.1GB/s data copy rate. This corresponds to an aggressive data copy rate using an SSE-enhanced memory copy routine when copying data from a cacheable memory source to a destination region marked as uncacheable, write-combining memory. The Intel Core 2 Duo processor features special write-combining buffers that allow aggressive burst mode transfers when copying from cacheable memory to write-combining memory. Due to the lack of shared virtual memory, the inter-shred communication between the IA shred and GMA X3000 shreds resembles that of traditional message passing communication between processes from different address spaces.

The second configuration, *Non-CC Shared*, assumes a shared virtual address space but without cache coherency between the IA sequencer an the GMA X3000 exo-sequencers. Data copying can be avoided in this case as both the IA sequencer and GMA X3000 exo-sequencers can access the identical physical memory location for the same virtual address. Memory writes performed by the IA sequencer or the GMA X3000 exo-sequencers may not be visible to the other until after a cache flush operation, which forces any dirty cache lines to be written back to main memory. However, data communication can still be accomplished by passing a pointer to a shared data structure between the IA sequencer and a GMA X3000 exo-sequencer as long as cache flush operations are appropriately invoked. Due to the lack of cache coherence, the IA shred and the GMA X3000 shreds need to use critical sections to enforce mutually exclusive access to shared data structures. The semaphore on the critical section will not be released until the GMA X3000 exo-sequencers completely flush the dirty lines to memory.

The first configuration, *Data Copy*, assumes a model without shared virtual memory and no cache coherence between the IA sequencer and the GMA X3000 exo-sequencers. Consequently, data communication between IA shred and GMA X3000 shreds requires explicit data copying, for which we assume a 3.1GB/s data copy rate. This corresponds to an aggressive data copy rate using an SSE-enhanced memory copy routine when copying data from a cacheable memory source to a destination region marked as uncacheable, write-combining memory. The Intel Core 2 Duo processor features special write-combining buffers that allow aggressive burst mode transfers when copying from cacheable memory to write-combining memory. Due to the lack of shared virtual memory, the inter-shred communication between the IA shred and GMA X3000 shreds resembles that of traditional message passing communication between processes from different address spaces.

The third configuration, *CC Shared*, models a cache-coherent shared virtual address space, which is the configuration assumed in Figure 10. In this model, data communication between the IA shred and the GMA X3000 shreds becomes much more efficient. Similarly, the synchronization on mutual access to shared data structure is also made much easier for programmers. For example, while critical sections are still necessary to provide mutual exclusion on writes to a shared variable, one shred can always read the shared variables that are updated by the other shreds. This allows more execution concurrency between shreds.

The performance data in Figure 11 demonstrate the benefits of a shared virtual address space compared to data copying. While significant performance improvement is still possible even with data copying, for computationally intensive kernels (e.g., `bicubic` and `ADVDI`), the gains are significantly reduced from the original *CC Shared* configuration in cases such as `LinearFilter` and `BOB`. For benchmarks in which the GMA X3000 performs little computation on the loaded input data, the time to copy data between separate address spaces represents a significant fraction of the processing time. Even with a highly optimized implementation on the latest IA Intel Core 2 Duo processor, the data copying achieves only 70.5% of that seen for a coherent shared virtual address space.

The cost of copying data can be ameliorated if the IA sequencer and the GMA X3000 exo-sequencers operate within a shared virtual address space, even if cache coherency is not supported. The time required to flush caches is still nontrivial, however, and the lack of coherency (*Non-CC Shared*) still yields 85.3% of the performance achieved with full cache coherency. Support for cache coherence improves performance because the cache flush operation is not needed to synchronize memory accesses.

For the *Non-CC Shared* configuration, when an IA shred spawns GMA X3000 shreds, it may appear necessary to flush the IA sequencer's cache fully before any GMA X3000 shred can be launched. In reality the majority of the cache flush operation on the IA sequencer can be overlapped with parallel shred execution on the GMA X3000 exo-sequencers if cache flush operations and shred launches can be interleaved. As each exo-sequencer shred only reads and writes a tiny portion of each data buffer (e.g., a 16 pixel by 16 pixel macroblock), as long as those data have been flushed back to memory by the IA producer shred, the exo-sequencer consumer shred for that macroblock can be launched and can execute safely. Additional cache flush operations can then proceed in parallel with useful work being performed in parallel on the exo-sequencers.

CONCLUSION

In this paper we present the EXO MIMD extension to the IA ISA to expose heterogeneous cores as application-level architecture resources and provide shared virtual memory to support the classic multi-shredded programming model for heterogeneous multi-core processors. The EXO architecture allows application programs to directly use heterogeneous hardware as MIMD functional units while requiring minimal additional dependency on the existing OS ecosystem. In addition, in order to take advantage of the rich ecosystem legacy for IA software development, the CHI programming environment provides an IA look-n-feel by

extending the Intel C++ Compiler, OpenMP runtime, and debugger toolchains to support user-level heterogeneous multi-shredding. Since its development, EXOCHI has been used in Intel's production media kernel development. Based on extensive feedback from developers, there is strong evidence that the IA look-n-feel of the programming environment has significantly improved productivity over prior device driver-based development environments.

ACKNOWLEDGMENTS

We thank Nick Yang, Porus Khajotia, Prasoonkumar Surti, Bob Dreyer, Sang-hee Lee, Katen Shah, Mike Dwyer, Yi-jen Chiu, Lian Tang, Igor Kozintsev, Xintian Wu, Bevin Brett, Susan Macchia, Ping Liu, Nenad Ukropina, Todd Schwartz, Jenny Nieh, David Sehr, Wei Li, and Sanjiv Shah for the productive collaboration throughout the EXOCHI project. We also appreciate the support from Shekhar Borkar, Joe Schutz, Tom Piazza, Justin Rattner, Jim Held, Steve Pawlowski, Kevin J. Smith, Bill Savage, Ketan Paranjape, Raj Hazra, Alan Crouch, Bryant Bigbee, Wilf Pinfold, Dave Shinsel, Ajay Bhatt, Doug Carmean, Per Hammarlund, Dion Rodgers, Steve Whalley, Avi Mendelson, and Prashant Sethi. In addition, we thank Anne Bracy, Ethan Schuchman, Ankur Khandelwal, Marian Lacey, and the anonymous reviewers whose valuable feedback has helped the authors greatly improve the quality of this paper.

REFERENCES

[1] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream Computing on Graphics Hardware," in *ACM Transactions on Graphics*, 23(3): 777–786, 2004.

[2] P. Dubey, "Recognition, Mining and Synthesis Moves Computers to the Era of Tera," *Technology@Intel Magazine*, February 2005.

[3] *GLSL–OpenGL Shading Language*, in www.wikipedia.org/wiki/GLSL*

[4] R. Gonzalez, "A Software-configurable Processor Architecture," *IEEE Micro*, Sept./Oct. 2006, pp. 42–51.

[5] *GPGPU: General Purpose Computation using Graphics Hardware*, at www.gpgpu.org*

[6] E. Grochowski, M. Annavaram, "Energy per Instruction Trends," in *Intel® Microprocessors. Technology@Intel Magazine*, March 2006, at http://www.intel.com/technology/magazine/research/energy-per-instruction-0306.pdf

[7] R. Hankins, G. Chinya, J. Collins, P. Wang, R. Rakvic, H. Wang and J. Shen, "Multiple Instruction Stream Processor, in *Proceedings of the 33rd International Symposium on Computer Architecture*, June 2006.

[8] J. Hoffman, D. A. Ilitzky, A. Chun, A. Chapyzhenka, "Architecture of Scalable Communication Core," in *First International Symposium on Networks-on-Chip*, 2007.

[9] Intel Corp., *Intel G965 Express Chipset*, at http://www.intel.com/products/chipsets/g965/prod_brief.pdf

[10] Intel Corp., "Intel's Next Generation Integrated Graphics Architecture – Intel Graphics Media Accelerator X3000 and 3000," *White Paper*, 2006.

[11] Intel Corp., "Tera-scale Research Prototype: Connecting 80 Simple Sores on a Single Test Chip," ftp://download.intel.com/research/platform/tera-scale/tera-scaleresearchprototypebackgrounder.pdf

[12] Intel Corp., "WiMAX," in *Intel Technology Journal* Vol. 8 Issue 3, at ftp://download.intel.com/technology/itj/2004/volume08issue03/vol8_iss03.pdf.

[13] U. Kapasi, S. Rixner, W. Dally, B. Khailany, J. Ahn, P. Mattson and J. Owens, "Programmable Stream Processors," in *IEEE Computer*, 2003.

[14] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas, "Single-ISA Heterogeneous Multi-Core Architectures for Multi-threaded Workload Performance," in *Proceedings of the 31st International Symposium on Computer Architecture*, June 2004.

[15] F. Labonte, P Mattson, W. Thies, I. Buck, C. Kozyrakis, and M. Horowitz, "The Stream Virtual Machine," in *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004.

[16] W. Mark, R. Glanville, K. Akeley, and M. Kilgard, "Cg: A System for Programming Graphics Hardware in a C-like Language," *ACM Transactions on Graphics* 22, 3, 896–907.

[17] M. McCool and S. Toit, *Metaprogramming GPUs with Sh*, A K Peters, Ltd., Wellesley, MA, 2004.

[18] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Eurographics*, August 2005.

[19] PeakStream Inc., "The PeakStream Platform: High Productivity Software Development for Multi-core Processors," *White Paper*, 2006.

[20] RapidMind Inc., "Performance Evaluation of GPUs using the RapidMind Development Platform," *Supercomputing'06*.

[21] S. Shah, G. Haab, P. Petersen, J. Throop, "Flexible control structures for parallelism in OpenMP," in

Proceedings of the First European Workshop on OpenMP, Sept. 1999.

[22] M. Segal and M. Peercy, "A Performance-Oriented Data Parallel Virtual Machine for GPUs," *SIGGRAPH*, 2006.

[23] E. Su, X. Tian, M. Girkar, G. Haab, S. Shah, and P. Petersen, "Compiler Support of the Workqueuing Execution Model for Intel SMP Architectures," in *EWOMP*, 2002.

[24] D. Tarditi, S. Puri, and J. Oglesby, "Accelerator: Using Data Parallelism to Program GPUs for General-Purpose Uses," in *Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2006.

[25] W. Thies, M. Karczmarek, and S. Amarasinghe,"StreamIt: A Language for Streaming Applications," *CC*, 2002.

[26] X. Tian, M. Girkar, S. Shah, D. Armstrong, E. Su, and P. Petersen, "Compiler and Runtime Support for Running OpenMP Programs on Pentium- and Itanium-Architectures," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, April 2003.

[27] O. Wechsler, "Inside Intel Core Microarchitecture: Setting New Standards for Energy-efficient Performance," *Technology@Intel Magazine*, 2006.

[28] D. Zhang, Z Li, H. Song, and L. Liu, "A Programming Model for an Embedded Media Processing Architecture," *SAMOS*, 2005.

AUTHORS' BIOGRAPHIES

Perry Wang is a Senior Staff Engineer with Intel's Corporate Technology Group. His work involves research on processor architecture, microarchitecture and compiler optimization techniques. Perry has been with Intel for 12 years and holds a master's degree in Computer Engineering from the University of Michigan.

Jamison Collins is a Staff Engineer with Intel's Corporate Technology Group. His work involves exploring and prototyping future Intel processor architecture and microarchitecture. Jamison has been with Intel for four years and holds a Ph.D. degree in Computer Science and Engineering from UC San Diego.

Gautham Chinya is a Senior Staff Engineer with Intel's Corporate Technology Group. His work involves exploring future processor system architecture and interaction with operating systems. Gautham has been with Intel for eight years and holds a master's degree in Computer Engineering from Purdue University.

Hong Jiang is a Senior Principal Engineer with Intel's Mobility Group. He is Intel's lead architect specializing in video technology. Hong has been with Intel for ten years and holds a Ph.D. degree in Electrical Engineering from the University of Illinois at Urbana-Champaign.

Xinmin Tian is a Principal Engineer with Intel's Software Solutions Group. He is Intel's lead compiler architect specializing in compiler parallelization, OpenMP, vectorization, and transactional memory compiler development projects. Xinmin has been with Intel for eight years and holds a Ph.D. degree in Computer Science from Tsinghua University.

Milind Girkar is a Principal Engineer with Intel's Software Solutions Group. He is Intel's lead compiler architect specializing in compiler parallelization and is responsible for planning the compiler requirements for future Intel processors. Milind has been with Intel for twelve years and holds a Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign.

Lisa Pearce is the software engineering manager with Intel's Mobility Group responsible for media development and content protection for all Intel integrated graphics solutions. Lisa has been with Intel for ten years and holds a bachelor's degree in Computer Science from Virginia Tech.

Guei-yuan Lueh is a Principal Engineer with Intel's Mobility Group. He leads the development of advanced compiler and runtime technology for Intel graphics solutions. Guei-yuan has been with Intel for 10 years and holds a Ph.D. degree in Computer Science from Carnegie Mellon University.

Sergey Yakoushkin is a Software Engineer in the Intel Corporate Technology Group. His work involves the development of software tools for emerging embedded platforms for communication acceleration, hardware-software co-design, and language design for data-streaming processing systems. Sergey has been with Intel for two years and holds an honours MS degree in Computer Science from St. Petersburg State University.

Hong Wang is a Senior Principal Engineer with Intel's Corporate Technology Group. His work involves research on future processor architecture and microarchitecture. Hong has been with Intel for twelve years and holds a Ph.D. degree in Electrical Engineering from the University of Rhode Island.

Package Technology to Address the Memory Bandwidth Challenge for Tera-scale Computing

Lesley Anne Polka, Assembly and Test Technology Development Division, Intel Corporation
Huthasana Kalyanam, Assembly and Test Technology Development Division, Intel Corporation
Grace Hu, Assembly and Test Technology Development Division, Intel Corporation
Satish Krishnamoorthy, Assembly and Test Technology Development Division, Intel Corporation

Index words: package technology, memory bandwidth, package design, electrical performance

ABSTRACT

Tera-scale computing stresses the platform architecture with memory bandwidth being a likely bottleneck to processor performance that presents unique challenges to CPU packaging. This paper describes the *evolution* in packaging technology with each processor generation to meet increasing memory bandwidth needs and the *revolution* in package technology required for tera-scale computing needs. The scope and focus of the paper are primarily design and electrical performance challenges. We discuss a potential roadmap of transitions in package architecture and technology that evolves from today's off-package memory scenario to increasingly complex on-package integrated memory architectures. An overall treatment of memory hierarchy, including off-die memory approaches, is not within the scope of this paper, but relevant to the overall challenge of enabling higher bandwidth. Again, the focus of this paper is on the CPU package itself. In this context, we discuss the memory bandwidth limitations, technology challenges, and tradeoffs of each package architecture.

INTRODUCTION

With a potential transition to tera-scale computing with multi- and many-core microprocessors and integrated memory controllers on the CPU, memory bandwidth becomes a bottleneck to processor performance [1]. This presents unique challenges to CPU packaging. Previous memory bandwidth requirements have scaled steadily, but fairly slowly, from one microprocessor generation to the next. This has driven a fairly steady but slow increase in pin count growth for chipset packages, which have traditionally provided the link to system memory between the microprocessor and memory modules. With a transition to multi- and many-core architectures, however, there is a large increase in the memory bandwidth requirement. This transition occurs at the same time as a shift to an integrated memory controller architecture for the CPU. These fairly simultaneous architecture transitions result in a tremendous burden on CPU packaging requirements, driving pin count growth and driving up routing density due to the large increase in interconnects that must be routed from the CPU through the package to off-package memory modules.

In this paper we describe the evolution in packaging technology with each processor generation to meet increasing memory bandwidth needs. We focus on the revolution in package technology required for tera-scale computing needs. The scope and focus of this paper are primarily design and electrical performance challenges. We propose a roadmap of transitions in package architecture and technology that evolves from today's off-package memory to increasingly complex on-package integrated memory architectures. We discuss the memory bandwidth limitations, technology challenges, and tradeoffs of each package architecture.

In the first section of this paper we look at memory bandwidth fundamentals. Next, we review the past trends in memory bandwidth requirements and the package technology impact. We follow this with sections describing the memory bandwidth needs for tera-scale computing and the resulting package technology impact and response.

MEMORY BANDWIDTH FUNDAMENTALS

It is useful to review several fundamental concepts as an introduction to the topic of memory bandwidth. First, it is important to understand the definition of memory bandwidth, the key elements related to bandwidth, and the role that the package interconnect plays. Very basically, memory bandwidth is defined as the product of the

number of data bits in the memory bus and the speed of a single bit in the bus. This can be expressed as

BW = # of bits x bit rate **Eq. (1)**

For example, if a memory bus is 8 bits wide (or 1 byte wide) and each bit transmits data at 1Gb/s (gigabits per second), then the memory bandwidth is 1 byte (1B) x 1Gb/s, or 1GB/s. A more realistic example is that of a typical DDR2 bus that is 16 bytes (128 bits) wide and operating at 800Mb/s. The memory bandwidth of that bus is 16 bytes x 800Mb/s, which is 12.8GB/s.

Besides the actual memory bandwidth, other key elements of memory bandwidth are latency and capacity. Latency is the roundtrip time that it takes to receive a response after a request has been sent. Latency is typically measured in nanoseconds (ns). Capacity refers to the size of the memory and is typically measured in MBs.

The memory subsystem hierarchy of a computer architecture consists of many levels. Memory can be located at the chip level, the package level, the board level, and in separate devices off the board (such as the hard disk). There is a tradeoff among the types and the key elements of memory (bandwidth, latency, and capacity) depending upon the location in the memory subsystem hierarchy. Very simply, faster, lower capacity memory is typically located on-chip, while slower, higher capacity memory is located off-chip. On-chip memory usually uses Static Random Access Memory (SRAM) technology, which is fast but expensive, and it is low-density compared to other memory technologies. On-chip memory usually serves as a cache and can be further divided into levels of cache, e.g., L1 cache, L2 cache, etc., [2]. Off-chip memory typically uses Dynamic Random Access Memory (DRAM) technology, which is slower but cheaper, and it is higher-density than SRAM. Off-chip memory located on the system board serves as the main memory for the computer system.

Today's typical computer architecture consists of the microprocessor (CPU), the chipset, and the main memory. Busses connect the various components of the system. Figure 1 illustrates a typical system architecture consisting of a microprocessor connected to a chipset through the system bus. The chipset in this example is divided into a Memory Controller Hub (MCH) and a separate Graphics Processing Unit (GPU). Each has a memory bus connecting to on-board memory. The system bus connects the CPU to the on-board, main system memory.

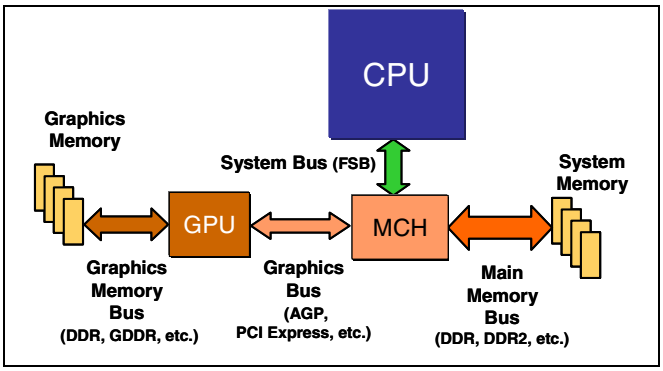


Figure 1: System architecture with main memory bus connected to the chipset

In this example, there are potential bottlenecks at each interconnect transition with respect to providing memory bandwidth to the CPU. Specifically, there is a transition from the CPU to the MCH through the system bus; and there is a transition from the MCH to the system memory through the main memory bus. The challenges to memory bandwidth in this traditional architecture have been to increasingly scale the capabilities of both the system and main memory busses to keep up with the steadily increasing memory demand of the CPU with each new generation. Figure 2 illustrates the historical trending of the system bus bandwidth capability vs. the memory bandwidth of the system. It makes sense that the two bandwidths have needed to scale simultaneously for optimum system performance. Scaling of bus capability has usually involved a combination of increasing the bus width while simultaneously increasing the bus speed.

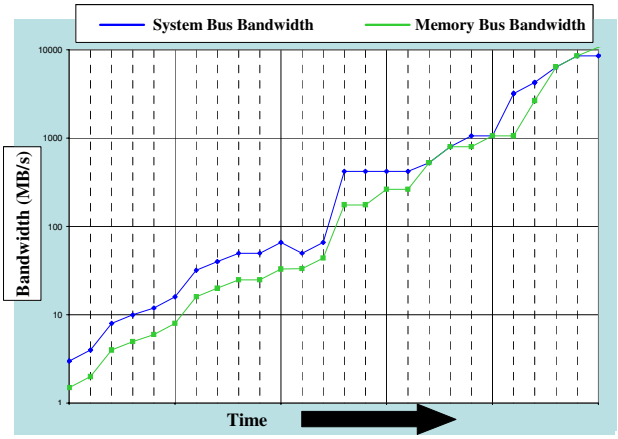


Figure 2: Historical trend for system bus bandwidth vs. memory bandwidth

Inherent in achieving both increased bus width and increased bus speed is the ability to scale the package technology to accommodate wider and faster busses. The package is the transitional interconnect between the fine pitch and high-density features of the processor chip and

the much coarser pitch and low-density features of the system board. The ability of packaging technology to serve as an intermediary interconnect bridging the gap between the chip and the system board has been critical to enabling increasing system memory bandwidth in the past. Packaging technology will continue to play a critical and increasingly important enabling role as we transition to tera-scale computing architectures.

REVIEW OF PACKAGE TECHNOLOGY EVOLUTION VS. MEMORY BANDWIDTH REQUIREMENTS

In the traditional system architecture of Figure 1, the packaging challenges associated with ever-evolving and increasing memory bandwidth impacted both chipset and CPU packaging. The chipset package has usually absorbed the need for increasing numbers of connections to system memory, while the CPU packaging dealt primarily with the need for an interconnect that could support increases in the system bus speed. Two almost simultaneous system architecture transitions are collapsing the focus of the memory bandwidth packaging challenges to primarily CPU packaging.

The first transition is the transition to an Integrated Memory Controller (IMC) in the CPU. Figure 3 illustrates the shift in the system architecture from that of Figure 1. The system memory now interfaces directly to the CPU through a system memory bus. The entire burden of pin count and interconnect speed to sustain increases in memory bandwidth requirements now falls on the CPU package alone.

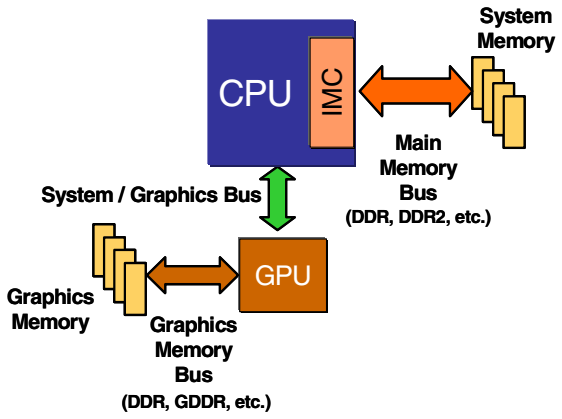


Figure 3: System architecture with integrated memory controller on the CPU

The second system architecture transition is the potential move to multi- and many-core CPU architectures. With the increase in cores comes a dramatic transition in memory bandwidth to "feed" the cores, particularly for the class of parallel applications envisioned. Since the CPU package is now the primary interconnect to the system memory, the CPU package bears the burden of the increase in memory bandwidth. Figures 4 and 5 illustrate the transition in memory bandwidth requirements and the projected CPU package pin count growth, respectively, as the transition to multi- and many-core CPUs with integrated memory controllers occurs.

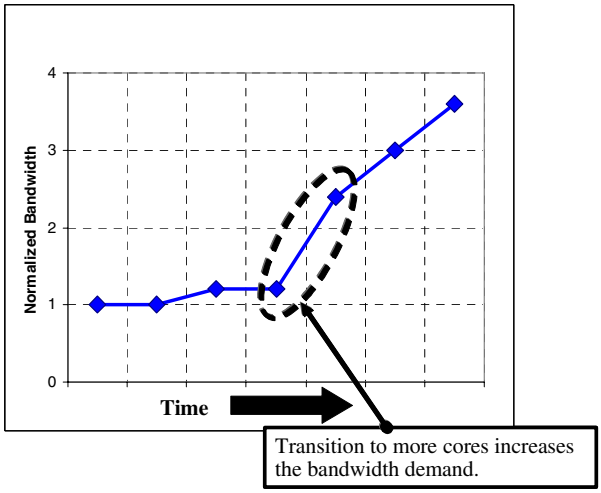


Figure 4: Bandwidth requirements

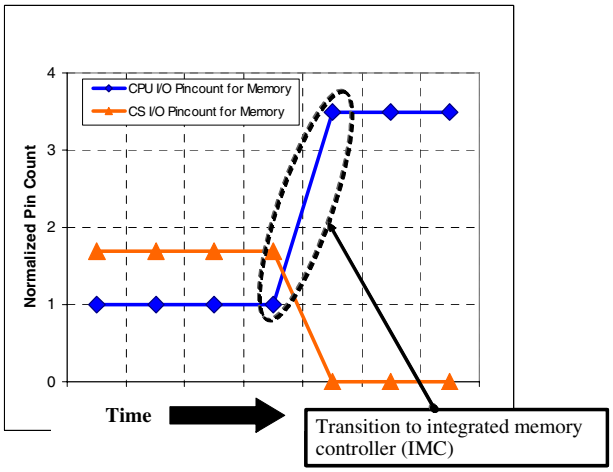


Figure 5: Package pin count growth for memory I/Os

One solution to address the bandwidth challenge and to stave off the continued increase in package pin count is to incorporate memory into a CPU + Memory Multichip Package (MCP). This changes the paradigm of packaging and the role of packaging in the memory hierarchy. Whereas packaging has previously been an enabler of higher bandwidths, now packaging would become a crucial sub-level in the overall memory system hierarchy. Moving to on-package memory, however, is not a trivial solution to implement. In the final main section of this

paper, we discuss the challenges and benefits/capabilities of an MCP configuration and various MCP architectures.

First it is beneficial to discuss tera-scale computing challenges in the area of memory bandwidth.

TERA-SCALE COMPUTING MEMORY BANDWIDTH CHALLENGES FOR PACKAGE TECHNOLOGY

Figure 6 shows the historical trend for memory bandwidth demand [3]. Today's bandwidth demand is in the 10–20GB/s range. From Figure 4 it is obvious that the move to multi- and many-core computing will easily drive a need for bandwidth in the 100s GB/s range in the not so distant future. Extrapolating from this, a target of 1TB/s of memory bandwidth for tera-scale computing architectures is not unreasonable [3–5].

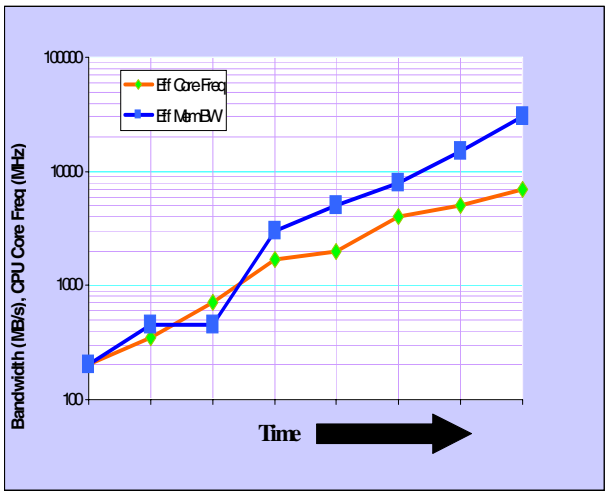


Figure 6: Historical trend for memory bandwidth [3]

Re-architecting a system capable of delivering this level of bandwidth is a challenge, given that the traditional methods are already reaching realistic limits. On many microprocessors, SRAM already occupies approximately half of the die real estate [4]. Increasing the amount of on-die memory becomes prohibitive from a cost and die size growth perspective. Increasing the bandwidth of board-level memory becomes prohibitive because of the continued increase in pin count and power with interconnect speed required to sustain bandwidth increases. Using on-package memory becomes a potential attractive intermediary level in the memory hierarchy that can work with chip-level and board-level memory to provide the bandwidth for tera-scale computing applications.

PACKAGE ARCHITECTURES TO MEET THE MEMORY BANDWIDTH NEEDS OF TERA-SCALE COMPUTING

To meet the memory bandwidth needs of tera-scale computing there needs to be an evolutionary transition in package architectures and technologies. In this section we discuss three architectures in detail: the CPU + memory 2D planar MCP, a package substrate embedded memory + CPU MCP architecture, and a 3D CPU + memory stacked die MCP. Figure 7 illustrates these MCP concepts.

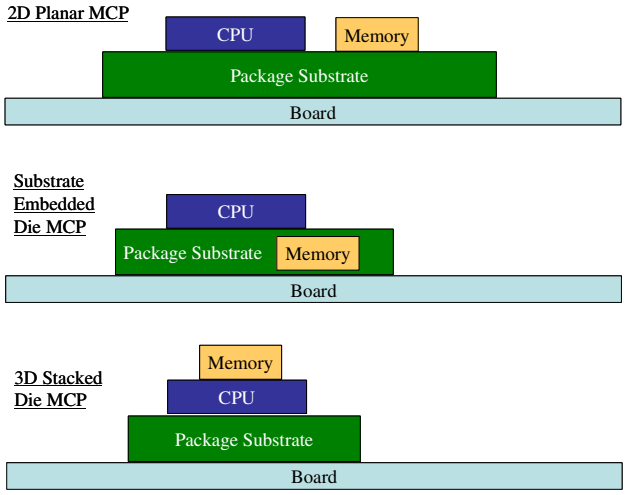


Figure 7: Memory + CPU package architectures for addressing bandwidth challenges

Each of these package architectures has benefits and challenges associated with the technologies. The memory + CPU 2D planar MCP is the most straightforward to implement and can be implemented with today's packaging technologies. There are capability limits on the amount of additional memory bandwidth this architecture can provide, however. Embedding memory in the package is the next evolutionary step in enabling higher memory bandwidth at the package level. This potentially enables higher bandwidths than the memory + CPU 2D planar MCP but requires technology development and comes with integration challenges. The final architecture, 3D stacked die memory + CPU MCP, potentially enables bandwidth capability surpassing the previous two architectures, but requires much technology development and has significant integration challenges. Given these tradeoffs among the architecture challenges and their bandwidth capabilities, a transitional package technology roadmap makes sense and is proposed in Figure 8.

For the remainder of this section we provide details on the capabilities and challenges of each of these on-package memory architectures. For the purposes of this discussion, we assume a memory technology that is capable of

delivering the bandwidth targets in terms of data rate and connection density that will be discussed. A discussion of the memory technology details is outside the scope of this paper. Also, the focus of our discussion is on memory bandwidth. Memory capacity and latency also play a key role in CPU system performance, but are outside the scope of this paper.

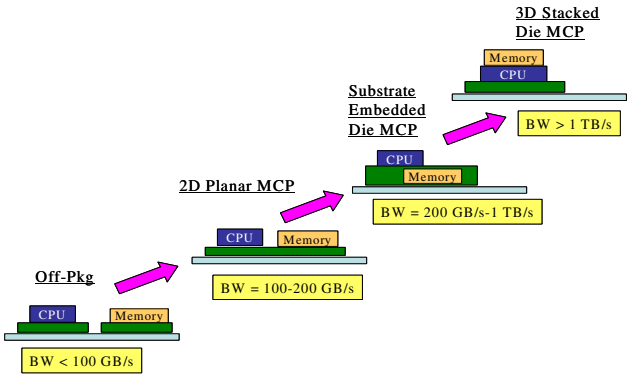


Figure 8: Proposed roadmap of package architecture transitions to address the memory bandwidth challenge

The first transition to on-package memory that can be implemented in the most evolutionary manner with respect to today's package technology is the memory + CPU 2D planar MCP. Intel has used 2D planar MCP packaging in the past and continues to use MCP package technology today for many of its multi-core processors, so this package technology is not a revolutionary technology. There are unique challenges associated with a CPU + memory 2D planar MCP that realistically limit its bandwidth capabilities.

The key challenges with a CPU + memory 2D planar MCP are that heterogeneous die are being assembled onto a single-package substrate with a requirement of optimizing performance to achieve a very fast, dense memory bus interconnection scheme. There are both design and electrical performance challenges associated with this architecture. Key design challenges are form factor fit, die placement, and routability.

In general, bump pitch between the CPU die and the memory or DRAM die will likely not match. This leads to routing issues that do not enable the design to take full advantage of line/space density capabilities of the package technology. Figure 9 illustrates a typical scenario when trying to route 200 I/Os between two die on an MCP substrate. Single-layer routing becomes impossible because of cut-off of the routing lanes. The solution is to use two-layer routing. This results in an increase in package cost and challenges in performance resulting from the division of the memory bus into two layers of routing.

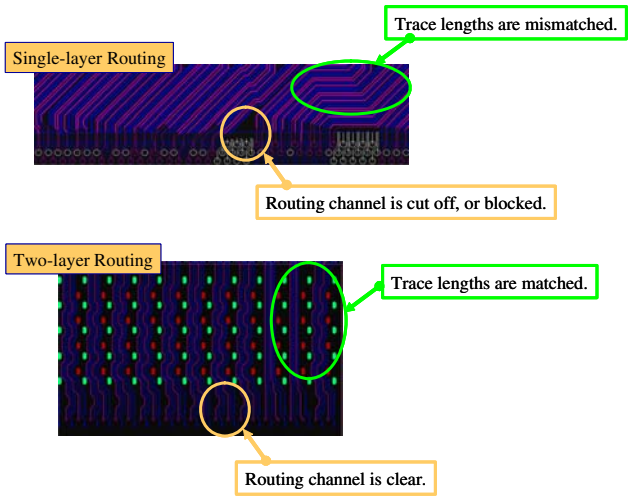


Figure 9: Design and routing challenges with on-package memory 2D planar MCP

In addition to routing challenges, there are challenges in large numbers of I/Os escaping from each die due to bump pitch constraints. Table 1 summarizes results for 200 and 400 I/Os. While routing a memory bus with 200 I/Os is fairly scalable with reasonable package technology and bump pitch routing capability scaling assumptions, increasing this to 400 I/Os becomes challenging.

Table 1: Design and routing/die-escape challenges with on-package memory 2D planar MCP

Attributes / Pitch	200 I/Os		400 I/Os	
Bump Pitch	150 um	175 um	150 um	175 um
Bumps / Row	72	62	72	62
# of Rows Needed (on-die)	3	4	6	7
I/O Region Depth (on-die)	0.5 mm	0.7 mm	0.9 mm	1.1 mm
Max. # of Lines Escapable thru Min. Pitch	2	3	2	3
Max. # of Rows Escapable	4	5	4	5
Pkg Layers Needed for Die Escape	1	1	2	2

Signal integrity issues in an MCP configuration also lead to performance challenges. Because there is still a substantial amount of trace length that is routed on the package between chips and these are routed very densely, crosstalk limits performance. For a single-ended configuration, the upper limit is ~6–7Gb/s. Results of a signal integrity sensitivity study are summarized in Figures 10 and 11.

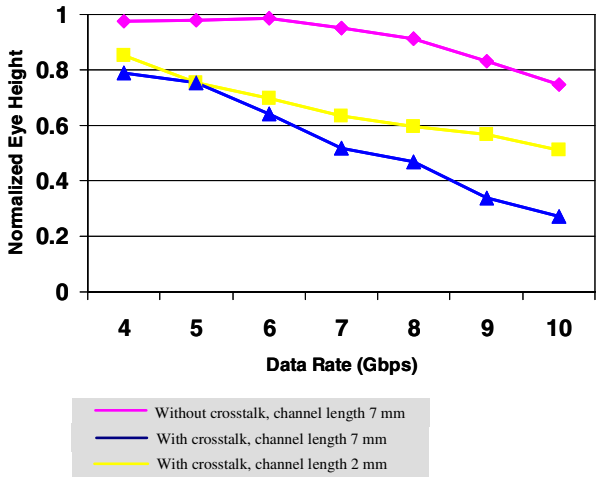


Figure 10: 2D planar MCP with on-package memory signal integrity results (eye height)

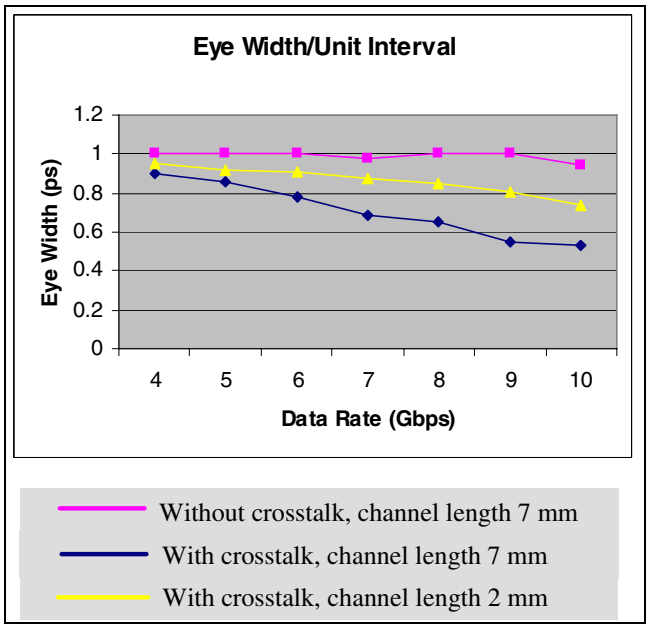


Figure 11: 2D planar MCP with on-package memory signal integrity results (eye width)

Combining the limits introduced by routing, fit, and signal integrity challenges, an estimate on the maximum sustainable bandwidth of a 2D planar MCP configuration is 100GB/s–200GB/s, depending upon the number of memory chips placed on the MCP. This also assumes a transition in memory technology to enable the types of connection densities and memory speeds used in this study. While this is a substantial amount of memory bandwidth capability, it is still not sufficient to meet the ultimate targets for tera-scale computing in the long term.

The next transition in package technology that can enable higher memory bandwidth than the CPU + memory MCP is a package embedded memory architecture. Figure 12 shows a schematic of this package architecture. This type of package architecture eliminates one level of transition between the CPU die and the DRAM die, i.e., the routing, which is responsible for the crosstalk that limits the ultimate I/O speed of the CPU + memory MCP architecture. This conceivably enables higher bandwidth by providing a very short and direct CPU-to-DRAM interconnect.

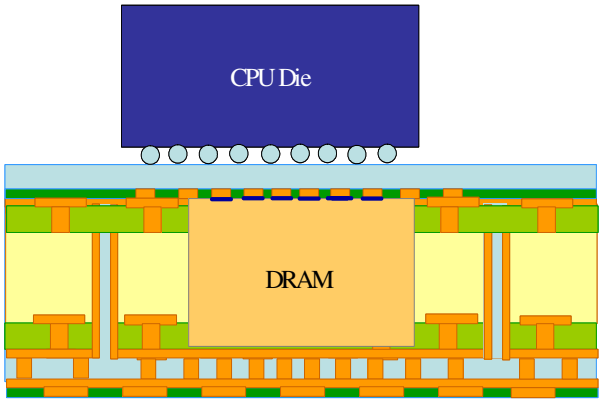


Figure 12: MCP with substrate embedded DRAM

Signal integrity simulations for a typical CPU-to-DRAM interconnect using a substrate embedded DRAM revealed very minimal impact due to crosstalk since the die-to-die connections are separated by an appreciable distance, equal to at least the bump pitch. The model used for the signal integrity studies is shown in Figure 13. At 4Gb/s, the substrate embedded memory configuration resulted in at least 30–40% more margin than the CPU + memory 2D planar MCP configuration results shown in Figure 10. Extrapolating from these results, it is conceivable that the substrate embedded memory architecture can easily achieve a bit rate of at least 10Gb/s. Given that direct connections between the CPU and memory can be made at the same pitch as the die bump pitch, hundreds of connections can be enabled in a small area. Consequently, this architecture can enable a memory bandwidth in the 200GB/s–1TB/s range.

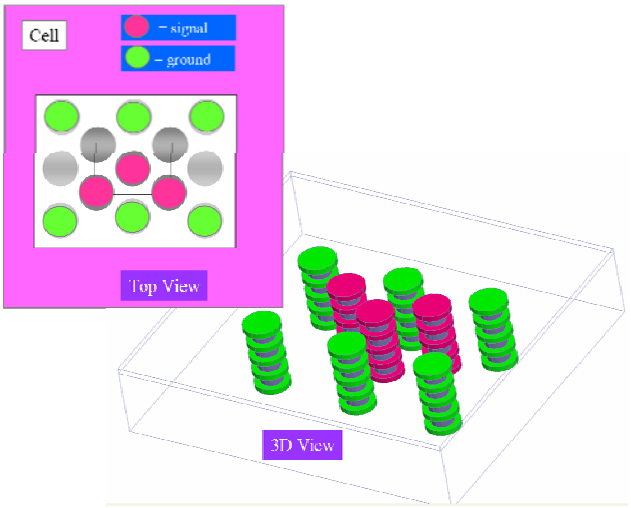


Figure 13: MCP with substrate embedded DRAM

One challenge with this type of architecture is the thermal performance with an embedded die. Preliminary thermal modeling and historical data suggest that a limit of

approximately ten watts or less for the embedded device should be maintained to avoid excessive refresh rates and increased power penalties. There are also substantial integration challenges with this architecture. This is, however, an intriguing architecture for enabling bandwidths in the 200GB/s-1TB/s range.

To enable memory bandwidths beyond 1TB/s, the 3D CPU + memory stacked die MCP architecture becomes interesting. Because this will provide the shortest possible interconnect between the CPU and memory die, the bit rate will far exceed 10Gb/s. In addition, the interconnect density will scale to enable thousands of die-to-die interconnects. Intel recently demonstrated a single-chip teraflop processor, Polaris, with 80 cores. Polaris contains hooks for stacking a separate SRAM chip, Freya, in a 3D configuration [6] and [7].

SUMMARY AND CONCLUSIONS

Memory bandwidth is one of the key challenges associated with tera-scale computing. Package technology will play a key role in answering that challenge. In this paper, we reviewed the historical trends in memory bandwidth and their impact on package technology. Key challenges in the past have been a continuing pin count growth that leads to package body size growth as well as design challenges. In addition, there are fundamental limits to the off-package memory bus speed that can be supported. A transition to on-package memory is necessary for supporting tera-scale computing needs. We reviewed a transitional, evolutionary roadmap for package technology to implement on-package memory architectures capable of meeting the needs of tera-scale computing. Table 2 summarizes the technology and architecture options, features, capabilities, and limits of each.

Table 2: Summary of features and capabilities of on-package memory architectures

	On-Die	On-Package			Off-Pkg
		2D Planar MCP	Substrate Embedded Die MCP	3D Stacked Die MCP	
		CPU Memory	CPU Memory	Memory CPU	CPU Memory
I/O Speed		4 – 7 Gb/s	4 - 10 Gb/s	10+ Gb/s	4 Gb/s
# of Connections		200 - 400	400 – 10,000	10,000+	~4 – 6 channels DDR
Bandwidth		100 - 200 GB/s	200 GB/s – 1 TB/s	> 1 TB/s	< 100 GB/s
Capacity	< 50 MB	512+ MB	< 512 MB	< 512 MB	> 1 GB
Interconnect Length	< 1 mm	~7 mm	1-3 mm	1 mm	> 100 mm

ACKNOWLEDGMENTS

The authors thank their colleagues who have worked with them on many aspects of this work, including Gans Ganesan, Tom Dory, Bhanu Jaiswal, Jemmy Sutanto, Jiangqi He, Anne Augustine, Ji Wu, Kinya Ichikawa, Yoshihiro Tomita, Sriram Dattaguru, Yonggang Li, Haluk Balkan, Ravi Mahajan, Debendra Mallik, Je-Young Chang, Greg Chrysler, Sriram Muthukumar, Jim Irvine, Islam Salama, Ihtesham Chowdhury, Aleksander Aleksov, and Henning Braunisch. We also acknowledge the continuing support of our managers: Kaladhar Radhakrishnan, Gaurang Choksi, Huong Do, Ken Brown, Bob Sankman, and Kemal Aygun. We thank the reviewers of this paper for their constructive input. Finally, we thank Jerry Bautista for serving as our mentor during the writing of this paper.

REFERENCES

[1] Held, J., Bautista, J. and Koehl, S., "From a Few Cores to Many: A Tera-scale Computing Research Overview," *Research at Intel White Paper*, download.intel.com/research/platform/terascale/terascale_overview_paper.pdf, 2006.

[2] Hsu, L., Iyer, R., Makineni, S., Reinhardt, S. and Newell, D., "Exploring the Cache Design Space for Large Scale CMPs," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, Sept. 2005, pp. 24–33.

[3] Mallik, D. Radhakrishnan, K., He, J., Chiu, C., Kamgaing, T., Searls, D. and Jackson, J. D., "Advanced Package Technologies for High-Performance Systems," *Intel Technology Journal*, vol. 9, no. 4, Nov. 9, 2005, pp. 259–271.

[4] Moore, S. K., "Winner: Masters of Memory," *IEEE Spectrum*, Jan. 2007, pp. 45–49.

[5] Patti, R., "Design and Application of 3D Memories," Proceedings of IMAPS International Conference and Exhibition on Device Packaging, Feb. 27, 2007.

[6] Vangal, S., et al., "An 80-Tile 1.28TFLOPS Network-on-Chip in 65 nm CMOS," in *Proceedings of ISSCC 2007 (IEEE International Solid-State Circuits Conference)*, Feb. 12, 2007.

[7] Johnson, R.C.., "Intel's Teraflops Chip Uses Mesh Architecture to Emulate Mainframe," *EE Times Online*, at www.eetimes.com/showArticle.jhtml?articleID=197004697, Feb. 12, 2007.

AUTHORS' BIOGRAPHIES

Lesley Anne Polka joined Intel in 1994. She is a Staff Electrical Packaging Engineer and works in the Assembly and Test Technology Development Division in Chandler, Arizona. Her focus is electrical issues related to development of Intel's packaging technologies for CPU, chipset, and wireless/RF packaging. During her twelve and a half years at Intel she has worked on various aspects of electrical packaging, including high-speed I/O and signal integrity, power delivery, analysis and characterization. She is presently working on new CPU package architectures to meet memory bandwidth and power-delivery challenges for future Intel microprocessors. Lesley has published over 15 papers on topics related to electrical design and technology development for microprocessor packaging and computational electromagnetics. She has two U.S. patents in packaging and a third U.S. patent pending in this area. Lesley obtained her B.S., M.S. and Ph.D. degrees, all in Electrical Engineering, from Arizona State University, in 1987, 1989 and 1995, respectively. Her e-mail is lesley.a.polka at intel.com.

Huthasana (Huthas) Kalyanam has been with Intel Corporation for the past one and a half years. He works as a Packaging Engineer within the Design Group in the Assembly and Test Technology Development Division in Chandler, Arizona. His primary job responsibilities include designing packages that are optimized for cost, electrical performance, and form factor. He designs packages that cater to Intel's Server and Desktop market segments. His responsibilities also include designing test packages for Intel's next-generation path-finding efforts in Packaging and Silicon. He is currently working on designing packages for multichip designs. Huthas received his M.S. degree in Electrical Engineering with a focus on VLSI and Analog systems from the University of Colorado at Boulder in 2004, after which he worked for a year with Instec Inc. Boulder as a Product and Design Engineer for Instec's Embedded Systems. His e-mail is huthasana.kalyanam at intel.com.

Grace Hu joined Intel in 2005 as an Electrical Packaging Engineer. She works on high-speed I/O characterization for CPU and chipset packages. Her focus areas also include high-frequency measurement and metrology development, broad-band dielectric material characterization, and risk assessments for new package technologies. Grace obtained her B.S. degree from Shanghai Jiao Tong University and her M.S. degree from the University of Missouri-Rolla, both in Electrical Engineering in 1999 and 2002, respectively. Her e-mail is grace.hu at intel.com.

Satish Krishnamoorthy joined Intel in 2005 and works as an Electronic Packaging Engineer in the Assembly and Test Technology Development Division at Chandler, Arizona. His focus is in package electrical analysis and design with particular emphasis on power delivery and signal integrity for various CPU and chipset products. Satish has published over five papers in refereed journals in the area of electronic circuits. He obtained his B.S. degree in Electronics and Communication Engineering from Anna University and his M.S. degree in Electrical Engineering from Arizona State University, in 2003 and 2005, respectively. His e-mail is satish.krishnamoorthy at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

Runtime Environment for Tera-scale Platforms

Bratin Saha, Programming Systems Lab, CTG
Ali-Reza Adl-Tabatabai, Programming Systems Lab, CTG
Richard L. Hudson, Programming Systems Lab, CTG
Vijay Menon, Programming Systems Lab, CTG
Tatiana Shpeisman, Programming Systems Lab, CTG
Mohan Rajagopalan, Programming Systems Lab, CTG
Anwar Ghuloum, Programming Systems Lab, CTG
Eric Sprangle, Visual Computing Group, DEG
Anwar Rohillah, Visual Computing Group, DEG
Doug Carmean, Visual Computing Group, DEG

Index words: runtime environment, scalable user level services, tera-scale platform

ABSTRACT

This paper presents the design and implementation of a runtime environment for tera-scale platforms. System software stacks currently view tera-scale platforms as an "SMP (symmetric multiprocessor) on a die." We show that there are fundamental differences between tera-scale and SMP systems that require that the software (SW) stack be re-architected. In particular, the SW stack needs to provide (1) support for efficient fine-grain parallelism, (2) programmability enhancements such as transactional memory, and (3) support for heterogeneous platforms and applications.

We discuss the design and implementation of a Many-Core RunTime (McRT) environment—a prototype tera-scale runtime environment—and show how it addresses the challenges of a tera-scale runtime. We also present simulation results from a tera-scale simulator to show that McRT enables excellent scalability on tera-scale platforms.

INTRODUCTION

System software tends to view a tera-scale chip multiprocessor (hereafter called TS-CMP) as a large-scale "symmetric multiprocessor (SMP) on a die"; yet, tera-scale CMPs have several characteristics that are fundamentally different from those of SMPs. It is critical to address these differences in order to implement a scalable and effective software stack. In particular it is important for the software stack to support (1) efficient fine-grain parallelism, (2) new concurrency abstractions that make parallel programming easier, and (3) platform and application heterogeneity.

Supporting Fine-grain Parallelism

TS-CMP has a very different compute-to-cache ratio than a traditional SMP. A 32-way SMP system typically has more than 100 MBs of aggregate cache size, while a 32-core TS-CMP has less than 10 MBs of cache. Thus a TS-CMP application needs to be threaded at a much finer granularity to reduce its working set. For example, MPEG4 encoding could be parallelized on a large-way SMP by encoding several frames in parallel. On a TS-CMP the encoding of an individual frame needs to be parallelized since the platform will not be able to cache multiple high-definition frames. Finally, many tera-scale applications benefit from fine-grain nested data parallelism rather than from coarse-grain task parallelism.

On the other hand, a TS-CMP enables fine-grain parallelism since inter-core communication is much easier—core-core bandwidth is of the order of terabytes/sec as opposed to gigabytes/sec for an SMP, and core-core latency is in the low tens of cycles (say 20 cycles) as opposed to hundreds of cycles in an SMP. Moreover, the effective core-core latency is much smaller, since the high degree of threading in a TS-CMP core allows some other thread (within the same core) to fully utilize the core resources if one thread is blocked on a cache miss.

Supporting New Concurrency Abstractions

Due to their high cost, large-way SMP systems have been restricted to niche markets, running applications written by sophisticated programmers whereas TS-CMP processors are targeted at mainstream price points and will bring parallelism to the average programmer. The success of TS-CMP processors and the applications that run on them depends on mainstream programmers embracing parallelism aggressively. Thus, the system SW stack should include new higher-level concurrency abstractions that make it easier for the average programmer to deal with parallelism.

Supporting Heterogeneity

Unlike SMPs, a TS-CMP software stack must comprehend heterogeneity at multiple levels. At the application level, TS-CMP processors will run a more diverse set of applications because they are targeted at a much broader market. At the hardware level, the TS-CMP platform may be heterogeneous with a combination of high-performance scalar cores, an array of high-throughput cores, and fixed function units. The system software stack must comprehend this heterogeneity. It needs to support configurable policies, for example, configurable scheduling policies, to adapt to different applications, and it needs to schedule applications according to their hardware requirements.

In this paper we present the design and implementation of McRT, a runtime environment for tera-scale platforms. McRT provides a configurable runtime framework that addresses the key tera-scale runtime requirements in the following ways:

- Fine-grain parallelism: McRT implements a significant fraction of threading services such as thread creation, synchronization, memory management, etc. at the user level. It also provides efficient user-level abstractions such as futures that make it easier to program and extract fine-grain parallelism.
- Concurrency abstractions: McRT includes a high-performance transactional memory library that supports an *atomic* construct in both C/C++ and Java. Transactional memory [15] provides a number of software engineering benefits compared to locks for managing access to shared data.
- Heterogeneity: McRT supports a number of configurable runtime policies that can be adapted for a particular application. In addition, McRT also supports multiple scheduling domains. Different hardware (HW) units can be mapped to different scheduling domains, and applications can be scheduled independently within each domain.

We show McRT's scalability using media encoding and Recognition, Mining, Synthesis (RMS) applications [11] on a tera-scale simulator. The results show that McRT's efficient threading primitives enable the applications to scale almost linearly up to 64 HW threads. We show that transactional memory can significantly ease parallel programming. Applications can use coarse-grain atomic blocks to synchronize access to shared data; yet they can achieve the performance of fine-grain locking. We also show a prototype implementation of a heterogeneous HW platform that leverages the support for scheduling domains in McRT.

McRT ARCHITECTURE

At its core, McRT contains a set of user-level threading primitives including a scheduler, memory manager, synchronization primitives, and a set of threading abstractions. We implemented these traditional operating system (OS) services as user-level primitives to improve efficiency by avoiding the expensive transitions between the user level and OS level making fine-grain parallelism more tractable. The McRT architecture is shown in Figure 1.

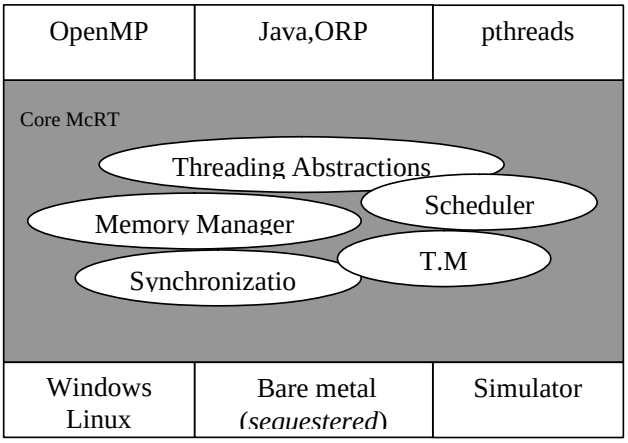


Figure 1: McRT architecture

McRT provides two user-level threading abstractions, threads and futures. The threads are similar to POSIX threads in functionality, while the futures are more lightweight and intended to support a concurrency idiom found in some languages such as MultiLisp [14] and CILK [8]. Futures provide a serial execution semantic, but can be executed in parallel if there are additional hardware resources.

The user-level scheduler is implemented as a task queue. An application can configure the number of task queues, e.g., specifying a single task queue for each processor. The application can also specify the scheduling policy, e.g., it can ask for a work-sharing policy where new tasks

are distributed among the task queues, or a work-stealing policy where idle processors search different queues for the next available task.

McRT uses a cooperative scheduling policy as opposed to the preemptive scheduling policy used predominantly in software stacks for SMPs. In an SMP system, the processing resource is expensive. Therefore the system software tries to timeshare the processing resource across multiple application threads by using preemptive scheduling. In a TS-CMP platform (say a platform with 128 cores), the processing resource is both inexpensive and abundant, which led us to use cooperative scheduling. This in turn addresses scalability bottlenecks, such as convoying, since an application can control when a thread gets preempted.

McRT includes a user-level synchronization library that includes different scalable algorithms such as MCS [24] locks and CLH [22] queues. It also includes a user-level memory allocator [16] that uses per-thread private allocation blocks. The allocator uses a completely non-blocking implementation that allows it to scale even with large oversubscription where the number of software threads is much greater than the number of hardware processors.

Finally, McRT includes a number of client adaptors that translate existing popular paradigms such as OpenMP and pthreads to the core McRT API. The OpenMP adaptor implements the API used by the Intel® C compiler, while the pthreads adaptor translates the POSIX API.

The core services in McRT are modularized and can be used as standalone services. For example, the memory manager ships as part of the Threading Building Blocks, while the transactional memory module has been tightly integrated into several compilers including the Intel C compiler, the StarJIT compiler [1], and the Harmony JITtrino compiler [5].

Evaluating Support for Fine-Grain Parallelism

We used a number of micro-benchmarks to evaluate the efficiency of the McRT threading primitives and hence its support for fine-grain parallelism. Figure 2 shows the results: the first row compares the cost of creating 255 threads; the second row compares the cost of 1000 consecutive lock acquire and release operations; and the final row compares the cost of 1000 context switches. In each case the *gettimeofday()* system call was used for the measurements. All the experiments were run on a 2.8GHz Intel® Xeon® processor. Column 2 reports the measurements observed by using native threads on Linux* 2.4.9, while Column 3 reports the measurements from using native threads on RedHat Enterprise* Linux 2.6.9-22ELsmp (NPTL 0.60).

	Native threads on Linux 2.4.9 µsec	Native threads on Linux 2.6.9 µsec	McRT µsec
Thread create (255 iterations)	21294	8960	1841
Mutex lock/release (1000 iterations)	120	82	81
Context switch (1000 iterations)	2927	3600	748

Figure 2: Micro-benchmark evaluation

We also measured the scalability of our threading primitives. Figure 3 compares the cost of creating thousands of threads on McRT and on Linux (2.6.9). Note that the efficiency of thread creation in McRT does not degrade even with thousands of threads.

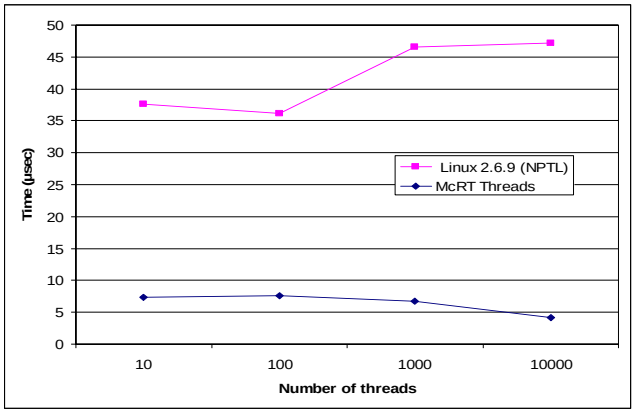


Figure 3: Scalability of thread creation

As mentioned before, McRT also implements futures to provide a lighter weight concurrency mechanism. Figure 4 compares the overhead of McRT futures to that of using McRT threads. For this, we created batches of futures and threads whose executable code simply returned immediately. We compared the time to complete such a batch using both threads and futures. Figure 4 compares the ratio of the execution time for threads and futures with futures being 40 to 100 times more efficient than threads. Obviously, futures can provide very good support for fine-grain parallelism.

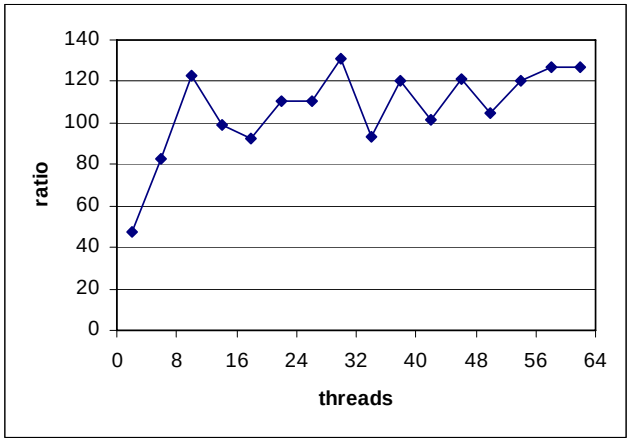


Figure 4: Thread vs. future creation overhead

NEW CONCURRENCY ABSTRACTION: TRANSACTIONAL MEMORY

Parallel programming poses many new challenges to the developer. A major one is synchronizing access to shared data between multiple threads. Traditionally, programmers have used locks for synchronization, but this method has well-known pitfalls such as deadlock and lack of composition. Transactional memory provides a new language construct that avoids the pitfalls of lock-based synchronization and eases concurrency control.

The McRT core services include a Transactional Memory (TM) library that supports the implementation of TM for C/C++ (unmanaged) and Java (managed). The TM library uses 2-phase locking to implement pessimistic write concurrency, and it uses versioning to implement optimistic read concurrency [27]. Every datum that may be accessed inside a transaction is associated with a transaction record—a pointer-sized word that mediates access to the shared datum. On a write, the TM library acquires exclusive ownership of the transaction record, performs an in-place update, and records the old value and the version number in an internal undo-log. On a read, the TM library records the version number of the transaction record (corresponding to the data address) in an internal read log. Before committing, the TM library validates a transaction by checking that the version numbers of the transaction records in the read set have not been changed. Upon committing, the lock is released and the version number is incremented. On an abort, the library uses the values in the undo-log to roll back the updates.

The TM library is also integrated with other runtime services such as memory management [16]. For example if a transaction allocates memory during its execution, the memory is automatically freed when the transaction aborts. Using a language-neutral API, we integrated the TM library with the Intel C/C++ compiler v10 and the StarJIT and JITtrino compilers for Java. These compilers take language-level *transactional* code blocks and insert calls to the appropriate runtime functions for every shared memory access inside the code block. This allows programmers to directly use TM rather than locks for concurrency control.

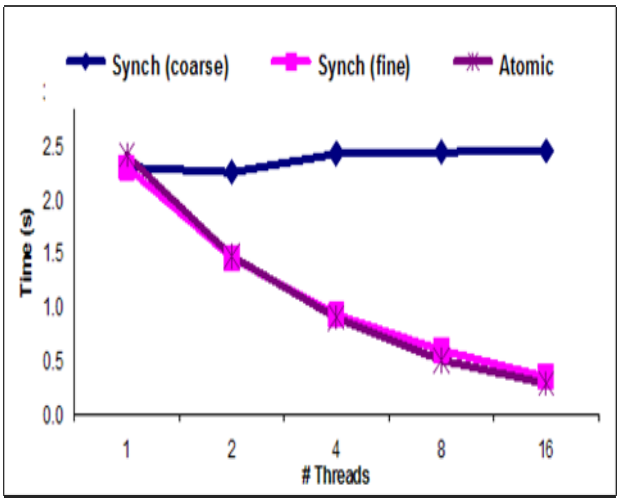


Figure 5: Transactions vs. locks on a hashmap

Figure 5 compares the performance of locks and transactions on a hashmap data structure. It measures the time taken to complete a set of insert, delete, and update operations on lock-based and transactional versions of the hashmap [2] on a 16-way SMP machine. The transactional version of the hashmap was obtained by replacing the critical sections in the coarse-grain synchronization version with atomic sections. As expected, coarse-grained locking (Sync(coarse)) does not scale, but both the transactional and the fine-grain version scale comparably, even though the transactional version uses coarse-grain synchronization.

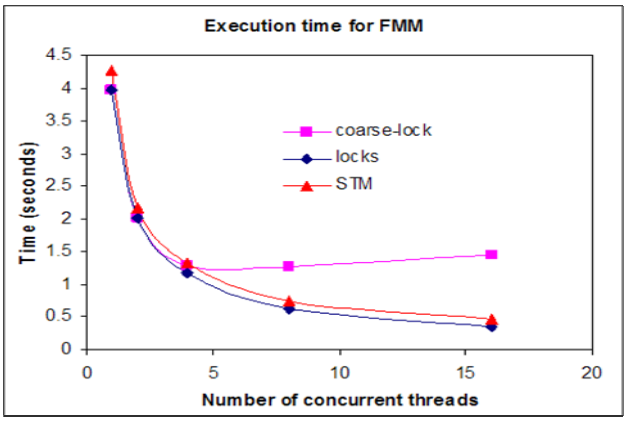


Figure 6: Transactions vs. locks on FMM (SPLASH2)

Figure 6 shows a similar result on the FMM benchmark, which is part of the SPLASH2 suite. The transactional version of the benchmark was obtained by replacing the coarse-grain critical sections with transactions. Again, the transactional version scales as well as the fine-grain version even though it uses coarse-grain synchronization.

These results show that McRT allows a programmer to leverage the software engineering benefits of transactional memory, such as composability and coarse-grain reasoning, while getting the performance of fine-grain locking. We expect this to be a key enabler in promoting multithreaded programming for tera-scale platforms.

SUPPORTING HETEROGENEITY

McRT uses multiple scheduling domains to support platform heterogeneity. Each domain can represent a set of hardware units with specific features such as good scalar performance, good throughput, special instruction sets, and so on. McRT extends the task queue mechanism to support scheduling domains. To create a domain D_k consisting of logical processors P_i to P_j a client creates a task queue Q_k that is accessed only by the processors P_i to P_j. New tasks created at these processors are only added to Q_k. The scheduler also exports an API that allows a task to yield its current logical processor and enqueue itself on a different task queue. A task executing in a domain D_k can switch to a different domain D'_k by enqueing itself on to the task queue Q'_k at which point it will get executed by the processors in D'_k . Applications, or even different parts of the same application, can be scheduled on different hardware units based on their requirements.

We prototyped a heterogeneous hardware platform on an 8-way SMP system. One processor (referred to as the OS processor) in the system boots up Windows* Server 2003, while the remaining seven processors (referred to as the sequestered processors) use McRT for all the threading services, without using the OS. The sequestered processors use a lightweight executive for interrupt handling. Thus, the sequestered processors emulate an attached compute engine, with the OS processor emulating a host CPU. Internally, McRT creates two scheduling domains, one representing the sequestered processors and the other representing the OS processor. The system configuration is shown in Figure 7.

We ran Equake* using the standard Spec input on the sequestered system. At the beginning the application is serial and reads the input. It then forks off a number of threads to perform the computation. McRT scheduled the serial part on the OS core and the parallel part on the sequestered cores.

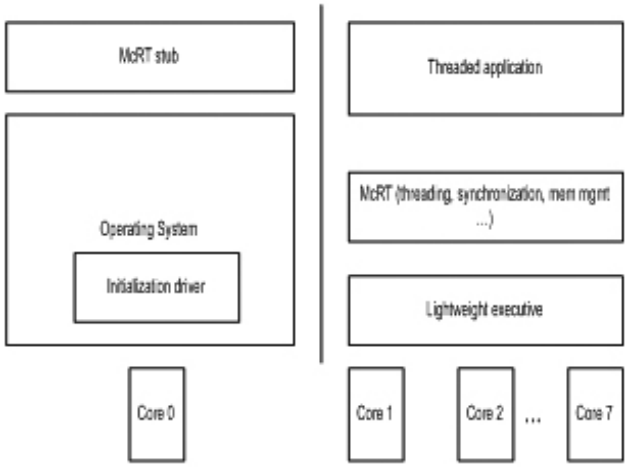


Figure 7: Sequestered system

Figure 8 shows the performance of Equake on the sequestered system. We first ran the benchmark on the 8-way SMP system with Windows running on all the processors. These numbers are reported as "Native" and "McRT-OS": "Native" refers to the performance from the Intel OpenMP* implementation (referred to as KAI in the figure), and "McRT-OS" refers to the performance from running McRT on top of Windows on the 8-way SMP. "McRT-Sequestered" refers to the performance on the sequestered system with one OS processor and seven sequestered processors. All speedups are reported with respect to the single thread "Native" execution time.

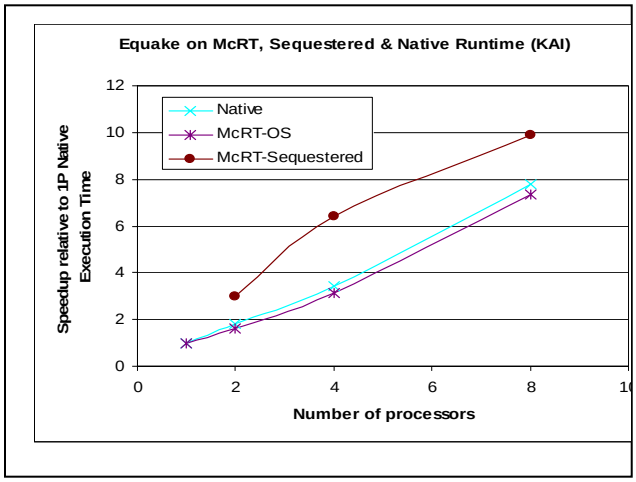


Figure 8: Equake on sequestered system

Equake performs much better on the sequestered system, mainly due to the fact that the software stack is much more lightweight and is not interrupted as often. Another reason is that in the sequestered mode, the application reserves and locks down enough memory at initialization so that it does not encounter page faults during execution.

RESULTS

We used a cycle-accurate simulator to evaluate McRT's performance on a TS-CMP processor. The simulated platform consists of an array of up to 16 in-order cores, each of which has four threads. Each core will select a different thread each cycle, round-robin, unless the thread is stalled due to, for example, a cache miss, being in the sleep state. The memory system consists of a 32 KB L1 data cache that is shared by all four threads in the core, a 2MB L2 cache that is shared by all the cores, and an off-chip 4MB L3 cache. All caches were simulated with an 8-way set associative configuration. The L1 cache access time is 3 cycles, the L2 cache access time is 12 cycles, and the L3 cache access time is 40 cycles. The simulator performs a cycle accurate simulation of the execution pipeline for all the HW threads, the different caches, the coherence protocol, the bandwidth for data transfer between different parts of the memory system, and the interconnect to the external memory.

We ported McRT to run directly on the simulator. Thus, the results reflect true execution driven simulation and accurately account for inter-thread synchronization. The simulator was modified to support system calls, while McRT provided all the threading services required by the application.

We used the popular open source MPEG4 encoder XviD (www.xvid.org*) and a set of RMS kernels [11] for Singular Value Decomposition (SVD) and Self Organizing Maps (SOM) as our workloads. The XviD encoder is used mainly on frames of 1920x1080 to correspond with frame sizes in emerging high-definition video. We show the performance for encoding the P frames since these (along with the B frames) happen to be the computationally intensive parts of the encoding. The simulated cache size does not allow multiple frames to be encoded in parallel; therefore, we had to parallelize the encoding of a single frame. A frame is partitioned into "k" sub-blocks, where "k" is the number of logical processors used for encoding. Thus, the scalability of MPEG4 encoding is a good test of the efficiency of McRT's fine-grain threading support.

SVD has numerous applications in the areas of data-mining and feature extraction, signal processing, and automated control; this workload uses the Jacobi method. An SOM is an unsupervised learning method represented by a two-layer neural network. Typically, it is used to map N dimensional data to two dimensions to discern patterns. It is extensively applied in text and feature mining, pattern recognition, and medical diagnostics.

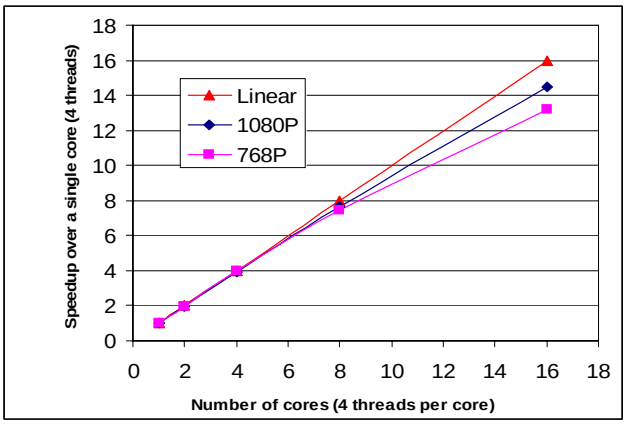


Figure 9: XviD speedup

Figure 9 shows the speedup of encoding a single frame as we increase the number of processors. The *x-axis* shows the number of cores. Note that for k cores, the number of HW processors is k*4. The graph uses the execution time on a single core (4 threads) as the baseline. Even at 16 cores (64 threads) we get almost a linear speedup (the "Linear" line in the graph represents speedup expected if the application was completely parallelized). The speedup on the 1080P (1920x1080) frame is slightly higher than on the 768P (1024x768) frame since the sub-block sizes are larger, and hence the cost of threading gets amortized.

Figure 10 shows the speedup for the RMS workloads. (The x-axis represents the number of cores, and the baseline is the execution time on a single core.) Both SVD and SOM scale almost perfectly up to 64 HW threads.

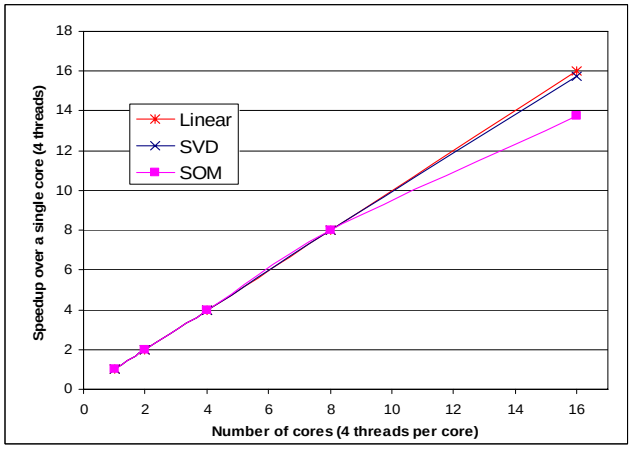


Figure 10: RMS speedup

RELATED WORK

Our work was inspired by previous projects in the areas of language systems, operating systems design, and high-performance computing. Several operating systems

have explored issues related to SMP scalability [3, 4, 17, 23, 26]. Disco [9] and K42 [18] have explored the design of scalable operating systems. Currently, operating systems such as Linux and MS Windows treat CMP architectures as an SMP on a chip. While the McRT framework arose out of our experience of these projects, this paper shows why we still need to rethink the system design to fully enable a scalable tera-scale environment.

The threading and synchronization primitives provided by McRT are similar to those seen in traditional user-level threading packages such as pthreads [19], NGPT [25], NPTL [10] and Capriccio [6]. The McRT scheduler is comparable to user-level schedulers that have been explored in the context of kernels such as L4 [20], Exokernel [12], Flux [13], and SPIN [7], but it is more lightweight and configurable. The McRT runtime is also comparable to language runtimes such as CILK [8] and OpenMP [21], but unlike these runtimes it is platform-neutral.

Finally, the specific mechanisms used in the McRT-STM [27] and the McRT memory manager [16] are described elsewhere, while the compiler integration is discussed in [2, 28].

CONCLUSION

A tera-scale platform has a number of fundamental differences from a traditional SMP system, which requires that the system software stack be redesigned to provide an effective and scalable runtime environment. In particular, the runtime environment must provide good support for fine-grain parallelism, support new concurrency abstractions that ease parallel programming, and support heterogeneous platforms and applications.

This paper described how McRT addresses the challenges of a many-core environment. To enable efficient fine-grain parallelism, McRT replaces many of the OS-level services with user-level primitives. Our results show that this enables a very scalable runtime stack that scales to more than 64 HW threads.

To ease parallel programming, McRT provides a high-performance TM library that supports a language-level *atomic* construct. TM provides several software engineering benefits compared to locks such as deadlock freedom, scalable composition, and failure atomicity. Additionally, our results show that transactions achieve the performance of fine-grain locking, yet allow coarse-grain synchronization.

McRT supports heterogeneity by dividing the platform into independent scheduling domains. These domains can be mapped to different hardware resources, and applications can be scheduled on the domain that best fits their requirements. In addition, McRT also supports a number of configurable runtime policies that allow it to adapt to different applications.

ACKNOWLEDGMENTS

We thank Leaf Petersen who provided the futures implementation in McRT and Cheng Wang who helped to implement the transactional memory support in the Intel C compiler. We also thank Jesse Fang who provided unstinted support and guidance for this work.

REFERENCES

[1] A. Adl-Tabatabai, J. Bharadwaj , D. Chen, A. Ghuloum, V. S, Menon, B. R. Murphy, M. Serrano, T. Shpeisman, "The StarJIT compiler: a dynamic compiler for managed runtime environments," *Intel Technology Journal*, Feb. 2003.

[2] A. Adl-Tabatabai, B.T. Lewis, V.S. Menon, B.M. Murphy, B. Saha, T. Shpeisman, "Compiler and runtime support for efficient software transactional memory," *PLDI*, 2006.

[3] T. E. Anderson, D. E. Lazowska, and H. M. Levy, "The Performance Implications of Thread Management Alternatives for Shared-Memory Multiprocessors," *IEEE Trans. on Comp*., Dec. 1989.

[4] T.E. Anderson, B.N. Bershad, E.D. Lazowska, and H.M. Levy, "Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism.,"*ACM ToCS*, Feb. 1992

[5] Apache Harmony Project at http://harmony.apache.org/*

[6] R. von Behren, J. Condit, F. Zhou, G. C. Necula, and E. Brewer, "Capriccio: Scalable threads for internet services," in *Proceedings SOSP-19*, 2003.

[7] B. N. Bershad, S. Savage, P. Pardyak, E. Gün Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, S. J. Eggers, "Extensibility, Safety and Performance in the SPIN Operating System," *SOSP*, 1995.

[8] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk, "An Efficient Multithreaded Runtime System," *PPoPP*, 1995.

[9] E. Bugnion, S. Devine, and M. Rosenblum, "Disco: running commodity operating systems on scalable multiprocessors," In *Proceedings SOSP-16*, 1997.

[10] U. Drepper and I. Molnar, "The native POSIX thread library for Linux, "January 2003, at http://people.redhat.com/drepper/nptl-design.pdf*.

[11] P. Dubey, "Recognition, Mining, and Synthesis moves computers to the era of tera," *Technology@Intel*, February 2005.

[12] D. R. Engler, M. F. Kaashoek, and J. O'Toole Jr., "Exokernel: an operating system architecture for application-specific resource management," *SOSP-15*, 1995.

[13] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers, "The Flux OSKit: A Substrate for Kernel and Language Research," *SOSP-16*, 1997.

[14] R. Halstead, "Multilisp: A Language for Concurrent Symbolic Computation," in *ACM Transactions on Programming Languages and Systems*, October 1985.

[15] M. Herlihy, and J. E. B. Moss, "Transactional memory: architectural support for lock-free data structures," *ISCA*, 1993.

[16] R. Hudson, B. Saha, A. Adl-Tabatabai, B. Hertzberg, "McRT-Malloc: A Scalable Transaction Aware Memory Allocator," *ISMM*, 2006.

[17] M. B. Jones, R. F. Rashid, "Mach and Matchmaker: Kernel and Language Support for Object-Oriented Distributed Systems," *OOPSLA*, 1986.

[18] The K42 project, IBM Research, at http://www.research.ibm.com/k42/*

[19] B. Lewis and D. J. Berg, *Multithreaded Programming with Pthreads*, Prentice Hall, New Jersey, 1998.

[20] J. Liedtke, "On micro-Kernel Construction," *SOSP-15*, 1995.

[21] H. Lu, Y. C. Hu, and W. Zwaenepoel, "OpenMP on networks of workstations," in *Supercomputing*, November 1998.

[22] P. Magnussen, A. Landin, and E. Hagersten, "Queue locks on cache coherent multiprocessors," *8th Intl. Parallel Processing Symposium*, Cancun, Mexico, April 1994.

[23] B. D. Marsh, M. L. Scott, T. J. LeBlanc, and E. P. Markatos, "Firstclass user-level threads," in *Proceedings SOSP-13*, October 1991.

[24] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," *ACM Transactions on Computer Systems*, 9(1):21–65, 1991.

[25] Next Generation POSIX Threading at http://www-124.ibm.com/pthreads/*

[26] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch, "The Sprite network operating system," *IEEE Computer*, 21(2):23–36, February 1988.

[27] B. Saha, A. Adl-Tabatabai, R. Hudson, C. Minh, B. Hertzberg, "McRT-STM: A High Performance Software Transactional Memory System For A Multi-Core Runtime," *PPoPP*, 2006.

[28] C. Wang, W. Chen, Y. Wu, B. Saha, A. Adl-Tabatabai, "Code Generation and Optimization for Transactional Memory Constructs in an Unmanaged Language," *CGO*, 2007.

AUTHORS' BIOGRAPHIES

Bratin Saha is a Senior Staff Researcher in Intel's Programming Systems Lab. His current research is focused on the design and implementation of modern concurrency abstractions, such as transactional memory, and highly concurrent runtime environments. He was one of the architects of locking and synchronization in the Nehalem processor. Bratin received his M.S. and Ph.D degrees in Computer Science from Yale University, and his B.S. degree in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur. His e-mail is bratin.saha at intel.com.

Ali-Reza Adl-Tabatabai is a Senior Principal Engineer in Intel's Programming Systems Lab. He leads a team of researchers working on compilers and scalable runtimes for future Intel® Architectures. Ali has spent most of his career building high-performance programming language implementations, including static and dynamic optimizing compilers and language runtime systems. His current research concentrates on language features, that make it easier for the mainstream developer to build reliable and scalable parallel programs for future multi-core architectures, and on architectural support for those features. Ali has published over 20 papers in leading conferences and journals. He received his Ph.D. degree in Computer Science from Carnegie Mellon University.

Richard L. Hudson is best known for his work in memory management including the invention of both the Train Algorithm and the Sapphire Algorithm. Richard joined Intel in 1998 where he has worked on concurrency related issues. He went to Shortridge and holds a B.A. degree from Hampshire College and an M.S. degree from the University of Massachusetts. His e-mail is rick.hudson at intel.com.

Vijay Menon is a Senior Research Scientist in the Programming Systems Lab at Intel investigating new programming technologies for multi-core systems. His primary areas of a research include compilers, managed runtimes, and transactional memory. Vijay holds a Ph.D.

degree in Computer Science from Cornell University and a B.S. degree in Electrical Engineering and Computer Science from the University of California at Berkeley.

Tatiana Shpeisman received her B.S. degree in Applied Mathematics from Leningrad Electrical Engineering Institute. She got her M.S. and Ph.D. degrees in Computer Science from the University of Maryland, College Park. Currently, she is a Senior Software Engineer working at the Intel Microprocessor Technology Lab. She is a member of ACM. Her e-mail is Tatiana.shpeisman at intel.com.

Mohan Rajagopalan's current research focuses on parallel runtime technologies for emerging many-core platforms. Mohan received his M.S. and Ph.D degrees from the University of Arizona in 2001 and 2006, respectively. His dissertation explored the application of language and compiler techniques to optimize overall systems design for automatically improving aspects such as security and dependability in addition to performance. His e-mail is mohan.rajagopalan at intel.com.

Anwar Ghuloum is a Principal Engineer with Intel's Microprocessor Technology Lab, working on diverse topics such as parallel language and compiler design, parallel architecture evaluation, optimizing memory system performance, and multimedia applications. Anwar received a B.S. degree in Computer Science and Engineering from the University of California, Los Angeles and a Ph.D. degree in Computer Science from Carnegie Mellon University's School of Computer Science in 1996. Before joining Intel, he co-founded and was the CTO of a fab-less semiconductor startup that designed parallel image and video processors for the consumer electronics market. Prior to that, Anwar developed novel predictive drug design software for early lead optimization using 3D surface pattern recognition techniques for a biotech startup. A recurring theme in Anwar's work has been to bridge high-level application knowledge and low-level parallel architecture constraints with careful parallel language and compiler design to achieve the optimal tradeoffs in productivity and performance. His e-mail is anwar.ghuloum at intel.com.

Eric Sprangle is a Principal Engineer with Intel's Visual Computing Group in Austin. Eric has been with Intel for eight years, working on the Intel® Pentium® 4 processor family, and he is currently one of the lead architects on the Larrabee project. Prior to joining Intel, Eric worked at ROSS Technology. Eric enjoys training for and racing in triathlons. His e-mail address is eric.sprangle at intel.com.

Anwar Rohillah is currently working as a VCG architect focusing on performance analysis and simulation. He has also worked on the Intel Pentium 4 processor family developing hardware prefetchers and doing performance analysis. Anwar obtained his B.A.Sc. degree in Computer Engineering from the University of Waterloo and joined Intel in 1999. His e-mail is anwar.rohillah at intel.com

Doug Carmean is a Senior Principal Engineer with Intel's Visual Computing Group in Oregon. Doug was one of the key architects responsible for definition of the Intel Pentium 4 processor. He has been with Intel for 18 years, working on IA-32 processors from the 80486 to the Intel Pentium 4 processor and beyond. Doug is currently the Larrabee Chief Architect. Prior to joining Intel, Doug worked at ROSS Technology, Sun Microsystems, Cypress Semiconductor and Lattice Semiconductor. Doug enjoys fast cars and scary, Italian motorcycles. His e-mail address is douglas.m.carmean at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

Architectural Support for Fine-Grained Parallelism on Multi-core Architectures

Sanjeev Kumar, Corporate Technology Group, Intel Corporation
Christopher J. Hughes, Corporate Technology Group, Intel Corporation
Anthony Nguyen, Corporate Technology Group, Intel Corporation

Index words: multi-core, loop and task parallelism, architectural support

ABSTRACT

In order to harness the additional compute resources of future Multi-core Architectures (MCAs) with many cores, applications must expose their thread-level parallelism to the hardware. One common approach to doing this is to decompose a program into parallel "tasks" and allow an underlying software layer to schedule these tasks on different threads. Software task scheduling can provide good parallel performance as long as tasks are large compared to the software overhead. We examine a set of Recognition, Mining, and Synthesis (RMS) applications and find that a significant number have small tasks for which software task schedulers achieve only limited parallel speedups.

We propose a hardware technique to accelerate dynamic task scheduling on MCAs with many cores. We compare this hardware to highly tuned software task schedulers for a set of RMS benchmarks with small tasks. The proposed hardware delivers significant performance improvements over the best software scheduler: for 64 cores, it is 88% faster on a set of loop-parallel benchmarks and 98% faster on a set of task-parallel benchmarks.

INTRODUCTION

Multi-core Architectures (MCAs) provide applications with an opportunity to achieve much higher performance than uniprocessor systems. Furthermore, the number of cores on MCAs is likely to continue growing, increasing the performance potential of MCAs. However, realizing this performance potential in an application requires the application to expose a significant amount of thread-level parallelism.

A common approach to exploiting thread-level parallelism is to decompose each parallel section into a set of tasks. At runtime, an underlying library or run-time environment distributes (schedules) these tasks to the software threads [2, 3, 4]. To achieve maximum performance, especially in systems with many cores, it is desirable to create many more tasks than cores and to dynamically schedule the tasks. This allows for much better load balancing across the cores.

We examine a set of benchmarks from an important emerging application domain: Recognition, Mining, and Synthesis (RMS) [1]. Many RMS applications have very high compute demands and can therefore benefit from a large amount of acceleration. Further, they often have abundant thread-level parallelism. Thus, they are excellent targets for running on MCAs with many cores.

For previously studied applications and architectures, the overhead of software dynamic task schedulers is small compared to the size of the tasks, and therefore, enables sufficient scalability. However, we find that a significant number of RMS applications are dominated by parallel sections with small tasks. These tasks can complete execution in as few as 50 processor clock cycles. For these, the overhead of software dynamic task scheduling is large enough to limit parallel speedups.

We therefore propose a hardware technique to accelerate dynamic task scheduling on scalable MCAs. It consists of two components: (1) a set of hardware queues that cache tasks and implement task scheduling policies, and (2) per-core *task prefetchers* that hide the latency of accessing these hardware queues. This hardware is relatively simple, scalable, and delivers performance close to optimal.

We compare our hardware proposal to highly tuned software task schedulers, and also to an idealized hardware implementation of a dynamic task scheduler (i.e., operations are instantaneous). On a set of RMS benchmarks with small tasks, it provides large performance benefits over the software schedulers and gives performance very similar to the idealized implementation.

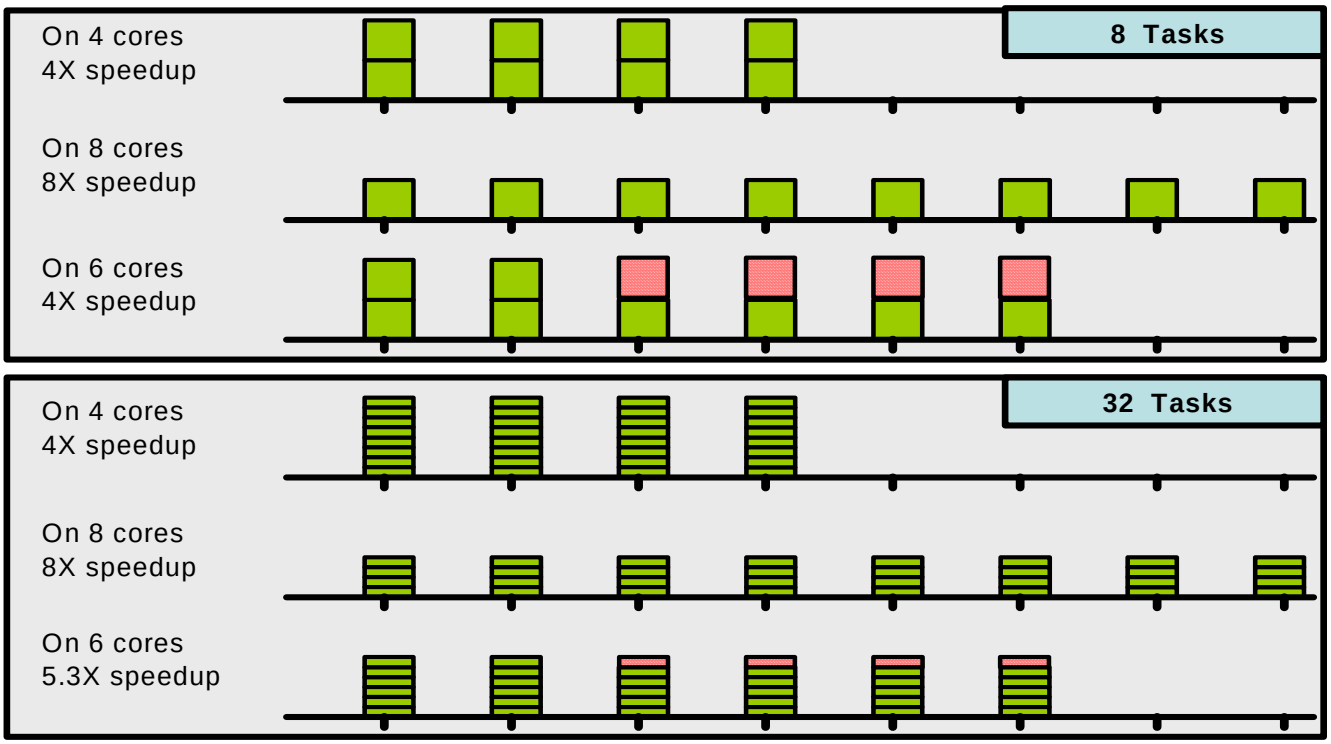


Figure 1: Impact of multiprogramming

Our contributions are as follows:

1. We make the case for efficient support for fine-grained parallelism on MCAs. Parallel tasks can be as fine as 50 processor clock cycles.
2. We propose a hardware scheme that provides architectural support for fine-grained parallelism. Our proposed solution has low hardware complexity and is fairly insensitive to access latency to the hardware queues.
3. We demonstrate that the proposed architectural support has significant performance benefits. First, it delivers much better performance than optimized software implementations: 88% and 98% faster on average for 64 cores on a set of loop-parallel and task-parallel RMS benchmarks, respectively. In addition, it delivers performance close to (about 3% on average) an idealized hardware implementation of a dynamic task scheduler (i.e., operations are instantaneous).

A CASE FOR FINE-GRAINED PARALLELISM

Previous work on dynamic load balancing targeted coarse-grained parallelism, i.e., parallel sections with either large tasks, a large number of tasks, or both. The target was primarily scientific applications for which this assumption is valid. For these applications, an optimized software implementation delivers good load balancing with an acceptable performance overhead.

The widespread trend towards an increasing number of cores becoming available on mainstream computers—both at homes and at server farms—motivates efficient support for fine-grained parallelism. Parallel applications for the mainstream are fundamentally different from parallel scientific applications that run on supercomputers and clusters in a number of aspects. We discuss these differences in detail in this section.

Architecture

Reduced communication overhead: MCAs dramatically reduce communication latency and increase bandwidth between cores. This allows parallelization of modules that could not previously be profitably parallelized.

Usage scenarios: These architectures are designed to be used with virtualization technologies as well as multiprogramming. In both these instances, the number of cores assigned to an application can change during the course of its execution. Maximizing the available parallelism under these conditions requires exploiting fine-grained parallelism.

Consider the example shown in Figure 1 that illustrates this using an 8-core MCA. It presents two scenarios where the parallel section is broken down into 8 and 32 equal-sized tasks (represented by green boxes). In a parallel section, if a core finishes its tasks before all other cores have finished their tasks, it has to wait. This results in wasted compute resources (shown in red). In each of the two scenarios, it shows the performance when varying number of cores are assigned to this parallel section. In both scenarios, with 4 and 8 cores, all the assigned cores are fully utilized. However, when 6 cores are assigned to the application, the first scenario wastes significant compute resources. In fact, it achieves the same speedup as when it was assigned 4 cores. In the second scenario, there are many fewer wasted compute resources because the parallel section was broken into finer-grained tasks.

This problem worsens when the number of cores increases. Figure 2 shows the maximum potential speedup on a 64-core MCA for a varying number of tasks. The ideal situation would be if the graph was linear, implying that each additional core would deliver additional performance. When only 64 tasks are used, the application would see no performance improvement even when the number of cores assigned to an application was increased from 32 to 63. To approach the ideal situation, one needs a much larger number of tasks (say 1024).

Performance portability across platforms: Parallel scientific computing applications are often optimized for a specific supercomputer to achieve the best possible performance. However, for mainstream parallel programs, it is much more important for the application to get good performance on a variety of platforms and configurations. This has a number of implications that require exposing parallelism at a finer granularity.

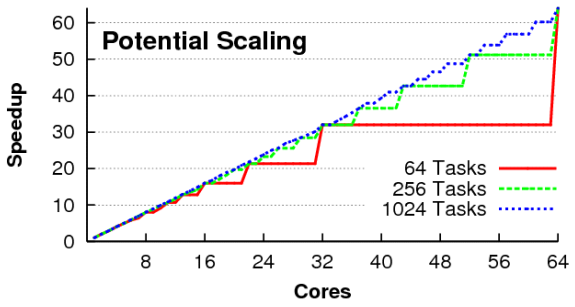


Figure 2: Theoretical scalability

First, the number of cores varies from platform to platform. For reasons similar to that for virtualization/multiprogramming, finer-granularity tasks are necessary.

Second, MCAs are likely to be asymmetric for a number of reasons including heterogeneous cores, Hyper-Threaded (HT) cores, Non-Uniform Cache Architecture (NUCA), and Non-Uniform Memory Architecture (NUMA). This means that the different threads on the core might progress at different rates. For instance, two threads sharing a core run at a different rate than two threads running on two different cores.

Figure 3 illustrates the impact of asymmetry with an example. Consider an application that breaks its parallel section into tasks that represent equal amounts of work (shown in green). However, asymmetry in architecture results in each task taking a different amount of time to complete. The result is wasted compute cycles (shown in red). This example shows that to ensure good performance in the presence of hardware asymmetry, it is best to expose parallelism at a fine grain.

Workloads

To understand emerging applications for multi-core architectures, we have parallelized and analyzed emerging applications (referred to as RMS [1]) from a wide range of areas including physical simulation for computer games as well as for movies, raytracing, computer vision, financial analytics, and image processing. These applications exhibit diverse characteristics. On the one hand, a number of modules in these applications have coarse-grained parallelism and are insensitive to a task queuing overhead. On the other hand, a significant number of modules have to be parallelized at a fine granularity to achieve reasonable performance scaling.

Recall that Amdahl's law dictates that the parallel scaling of an application is bounded by the serial portion. For instance, if 99% of an application is parallelized, the remaining 1% that is executed serially will limit the maximum scaling to around 39X on 64 threads.

This means that even small modules need to be parallelized to ensure good overall application scaling.

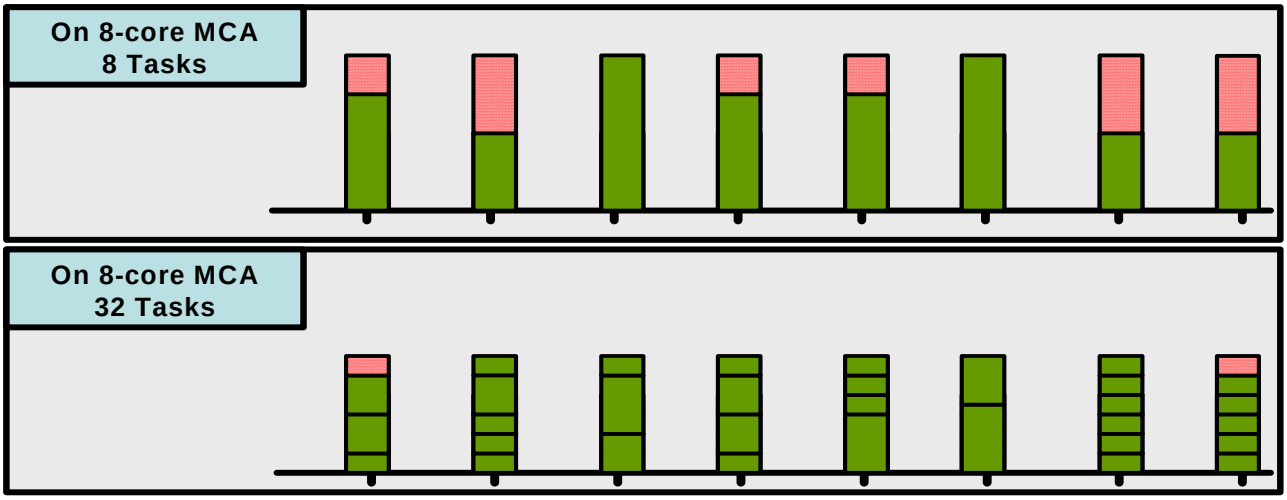


Figure 3: Impact of asymmetry in architecture

Ease of Programming

The use of modularity will continue to be very important for mainstream applications for several reasons. Modularity is essential to developing and maintaining complex software. In addition, applications are increasingly composed of software components from multiple vendors. These include middleware as well as libraries optimized for specific platforms.

Modular programs require writers of individual modules to make decisions about how best to parallelize that module. Consider a simple example where an application is composed of two modules: the main program and an optimized math library. Suppose that parallelizing either the library or the main program is sufficient to exploit all the parallel computing resources on the machine. However, modularity dictates that one module does not make assumptions about another module. This requires that for the best performance on a variety of platforms, both modules be parallelized in cases where the other module is not parallelized. The net result will be a finer granularity of parallelism in the application.

ARCHITECTURAL SUPPORT FOR FINE-GRAINED PARALLELISM

Software implementations of task queues incur an overhead (e.g., for enqueues and dequeues). This overhead grows with an increasing number of threads due to increased contention on shared data structures in the software implementation. Thus, if the tasks are small, the overhead can be a significant fraction of application execution time. This limits how fine-grained the tasks can be made and still achieve performance benefits with a large number of cores.

Therefore, we investigate adding hardware for MCAs that accelerates task queues. This hardware operates under the covers (i.e., is not visible to application writers) to accelerate the task queue operations that are key to high performance on many-core architectures. In particular, it provides very fast access to the storage for tasks. This includes performing fast task scheduling (i.e., determining which task a core should execute next). Its task scheduling is based on work stealing—a well-known scheduling algorithm.

Using the Proposed Hardware

Applications interface to the proposed hardware via a software library. This allows programmers to use the same intuitive API that they use for software implementations of task queues. Since the software library hides the proposed hardware from applications, only the library needs to directly interact with this hardware. Besides initialization and termination, the only operations that the library needs to perform are task enqueues and dequeues.

In current software task queue implementations, each task is represented as a tuple, a set of associated items, as shown in Figure 5. Typically, the tuple entries will be function pointers, jump labels, pointers to shared data, pointers to task-specific data, and iteration bounds, but they could be anything. An enqueue places a tuple into a software data structure for storage, and a dequeue retrieves a tuple from the data structure.

Our proposed hardware is primarily intended to accelerate enqueue and dequeue operations. Thus, the hardware stores tuples on enqueue operations and delivers tuples on dequeue operations. It does not interpret the contents of the tuples. This provides flexibility to the software library in that the library determines the meaning of each entry in a tuple. This flexibility allows the library writer to optimize for applications with different needs.

Our Proposed Hardware

We consider an MCA chip where the cores are connected to a cache hierarchy by an on-die network. The proposed hardware consists of two separate hardware components: a Local Task Unit (LTU) per core, and a single Global Task Unit (GTU). This is illustrated in Figure 4.

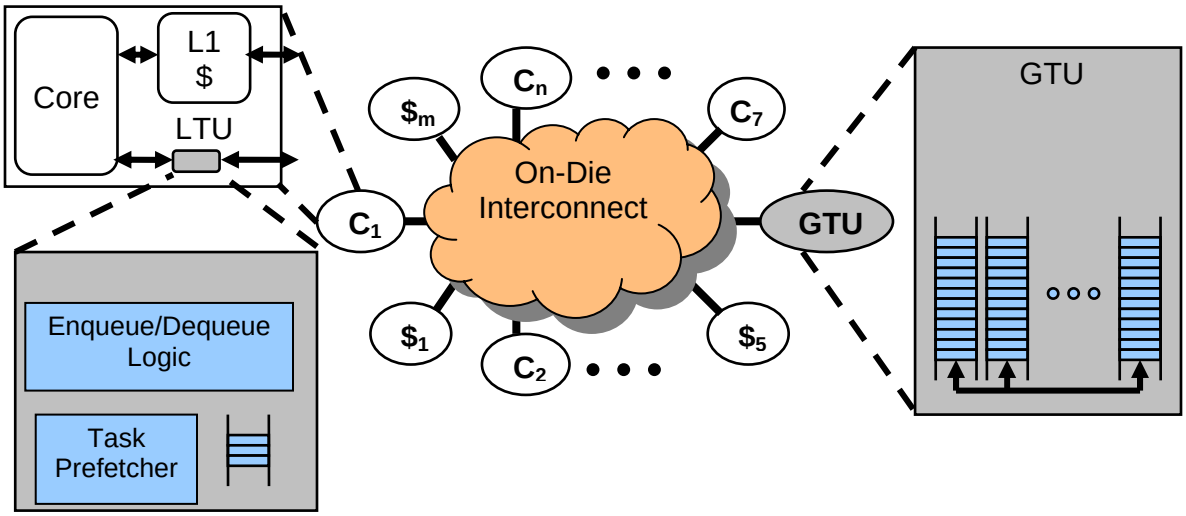


Figure 4: Our proposed hardware in a MCA chip with cores (C_i) and parts of the last-level shared cache ($\$_i$)

Global Task Unit (GTU)

The GTU holds enqueued tasks in a set of hardware queues. There is one hardware queue per logical core in the chip. This allows the use of the distributed task scheduling algorithm. The GTU also includes logic for implementing this algorithm. Since the hardware queues are physically close to each other, the proposed scheme can quickly determine which queues are empty and which are not. This makes stealing tasks much faster than for software implementations of distributed task scheduling. It also allows the hardware to quickly detect when all tasks are complete. This is important so that the main thread can start executing the serial code following the parallel section as quickly as possible.

Function pointer	Parameter 1	Parameter 2	Parameter 3

Figure 5: An example task tuple format

The GTU is physically centralized on the chip. Communication between the GTU and the cores is via the same on-die interconnect as the cache subsystem. The downside of a physically centralized GTU is that as the number of cores on a chip increases, the average communication latency between a core and the GTU also increases. This latency, if not hidden, could impact performance. Therefore, we address this with task prefetchers at each core, as described below.

The size of the queues in the GTU is bounded. When the queues are full, the hardware generates an exception. The exception handler can move some of the tasks from the hardware queues into memory creating room for future task enqueues. An underflow mechanism is used to move the overflown tasks back into hardware queues at a later point [2].

Multiprogramming is also supported by the hardware by using the same overflow and underflow mechanism to move tasks from hardware into memory, and vice versa, on context switches [2].

Local Task Unit (LTU)

Each core has a small piece of hardware to interface with the GTU, called the LTU. In addition to hardware for interfacing with the GTU, the LTU also contains a task prefetcher and small buffer to hide the latency of accessing the GTU. Hiding this latency can significantly improve performance. While a typical enqueue operation from a thread can be almost entirely overlapped with useful work, a dequeue operation is on a thread's critical path. If a logical core were to wait to contact the GTU

until the thread running on it finished its current task, the thread would have to stall for the entire GTU access latency. If the latency is significant compared to the size of a task, this can take up a significant fraction of an application's execution time. Therefore, the LTU tries to hide the dequeue latency. It does this by trying to keep at least one task in the LTU's buffer at all times. A dequeue is able to grab such a task very quickly. The LTU operates as follows.

- On a dequeue, if there is a task in the LTU's buffer, that task is returned to the thread and a prefetch for the next available task is sent to the GTU. When the GTU receives a prefetch request, it treats it as a regular dequeue, and will steal a task from another logical core's queue if necessary.
- On an enqueue, the task is placed in the LTU's buffer. Since the proposed hardware uses a LIFO ordering of tasks for a given thread, if the buffer is already full, the oldest task in the buffer is sent to the GTU.

In our experience, the LTU's buffer only needs to hold a single task to hide the GTU access latency. If this latency grows in the future, the buffer could be made larger. However, there is a cost: tasks in an LTU's buffer cannot be stolen since they are not visible to the GTU. This could hurt performance if there are only a few tasks available at a time.

Table 1: Loop-level benchmarks and their inputs

Benchmark	Data set
Gauss-Seidel	128x128, 256x256, 512x512
Dense Matrix-Matrix Multiply (MMM)	64x64, 128x128, 256x256
Dense Matrix-Vector Multiply (MVM)	64x64, 128x128, 256x256, 512x512
Sparse Matrix-Vector Multiply (MVM)	4 data sets
Scaled Vector Add	512, 1024, 4096, 16384 elements

Table 2: Task-level benchmarks and their inputs

Benchmark	Data set
Game Physics Constraint Solver	4 Models
Binomial Tree	512, 1024, 2048, 4096
Canny Edge Detection	cars, costumes, camera2, camera4
Cholesky Factorization	4 data sets
Forward Solve	4 data sets
Backward Solve	4 data sets

EXPERIMENTAL EVALUATION

Benchmarks

We evaluate our proposed hardware on benchmarks from a key emerging application domain: RMS. All benchmarks were parallelized within our lab. Table 1 and Table 2 give the benchmarks and their data sets.

Loop-level parallelism: We use primitive matrix operations and Gauss-Seidel as a set of benchmarks with loop-level parallelism since these are both very common in RMS applications and very useful for a wide range of problem sizes. Most of these benchmarks are standard operations and require little explanation. The sparse matrices are encoded in compressed row format. Gauss-Seidel iteratively solves a boundary value problem

with finite differencing using a red-black Gauss-Seidel algorithm. These benchmarks are straightforward to parallelize; each parallel loop simply specifies a range of indices and the granularity of tasks. We evaluate each benchmark with several problem sizes to show the sensitivity of performance to problem sizes.

Task-level parallelism: We use modules from full RMS applications as a set of benchmarks with task-level parallelism. Task-level parallelism is more general than loop-level parallelism where each parallel section starts with a set of initial tasks and any task may enqueue other tasks. These benchmarks represent a set of common modules across the RMS domain. Some of the benchmarks are based on publicly available code, and the remaining ones are based on well-known algorithms. These benchmarks are as follows:

1. The Binomial Tree uses a 1D binomial tree to price a single option. Given a tree of asset prices, the algorithm derives the value of an option at time 0 (that is, now) by starting at time T (that is, the expiration date) and iteratively stepping "backward" toward t=0 in a discrete number of time steps, N. At each time step t, it computes the corresponding value of the option Vi at each node of the tree i. The number of exposed tasks (i.e., nodes) is small at any time step and the task size is small. Hence, task queue overhead must be small to effectively exploit the available parallelism.
2. The Game Physics constraint solver iteratively solves a set of force equations in a game physics constraint solver. For inputs that have few bodies and constraints, the amount of parallelism is limited, especially for a large number of cores.
3. Cholesky, Backward Solve, and Forward Solve are operations on sparse matrices. Cholesky performs Cholesky factorization. Backward Solve and Forward Solve perform backward and forward triangular solve on a sparse matrix, respectively. These solvers use a data structure called elimination tree that encodes the dependency between tasks. The irregularity in sparse matrices results in high variation of task size. Therefore, we need very efficient task management to achieve good load balancing.
4. The Canny Edge Detection computes an edge mask for an image using the Canny edge detection algorithm. It first finds a group of edge candidates and determines whether their neighbors are likely to be edges. If so, the algorithm adds them to the candidate list and recursively checks their neighbors for candidacy. The number of tasks correlates directly to the number of candidates and tends to be small. Further, the amount of work in checking for candidacy is also very small.

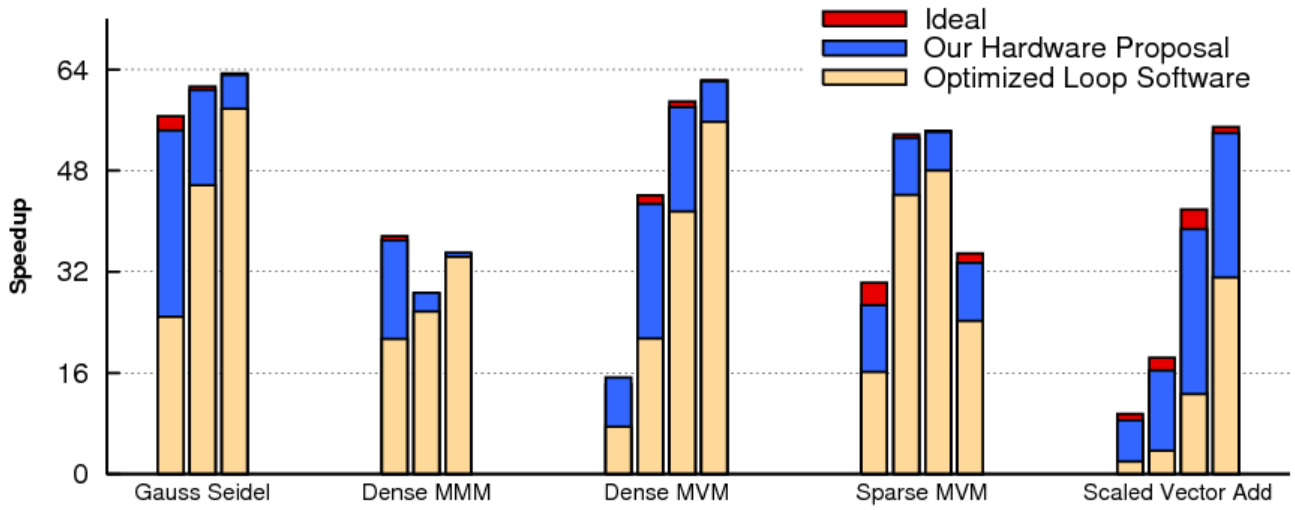


Figure 6: Performance of loop-level benchmarks

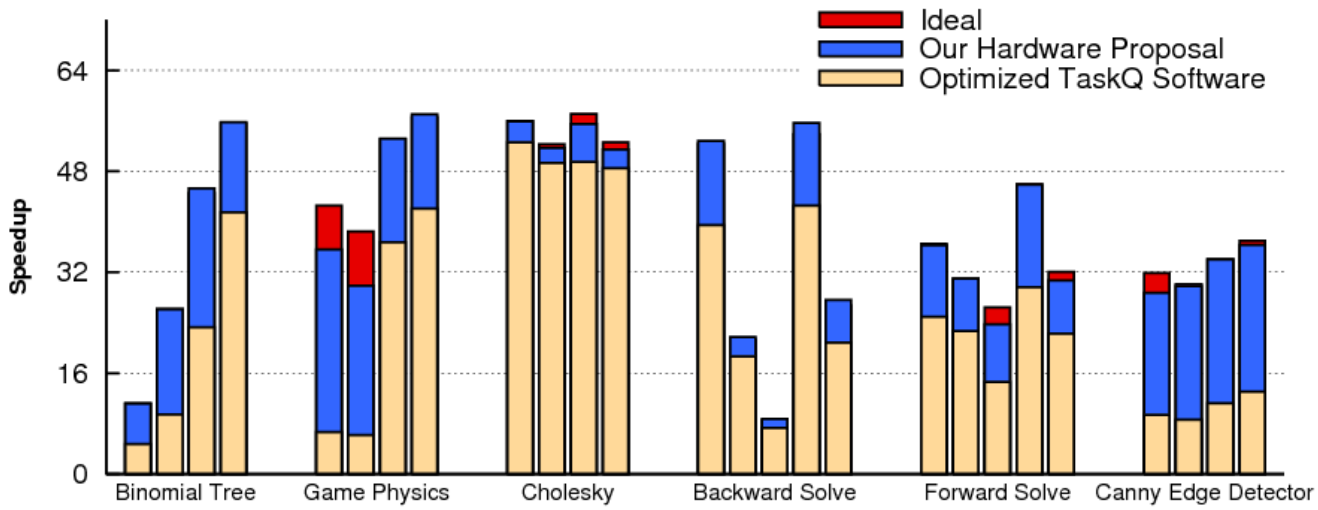


Figure 7: Performance of task-level benchmarks

Results

Figures 6 and 7 show the performance benefit of our proposed hardware for the loop-level and the task-level benchmarks, respectively, when running with 64 cores. In particular, the hardware proposal is compared with the best optimized software implementations and an idealized implementation (Ideal) in which tasks bypass the LTUs and are sent directly to/from GTU with zero interconnect latency. Additionally, the GTU processes these tasks instantly without any latency. The optimized software implementation uses a combination of widely used state of the art techniques [2, 3] that deliver the best performance.

The graphs represent the speedup over the one-thread execution using the Ideal implementation. For each benchmark, they show multiple bars. Each bar corresponds to a different data set shown in Tables 1 and 2.

For the loop-level benchmarks in Figure 6, the proposed hardware executes 88% faster on average than the optimized software implementation and only 3% slower than Ideal.

For the task-level benchmarks in Figure 7, on average the proposed hardware is 98% faster compared to the best software version and is within 2.7% of Ideal. For Game Physics with one data set, and Forward Solve with another data set, the amount of parallelism available is very limited. In the software implementations, the cores contend with each other to grab the few available tasks, which adversely impacts performance.

CONCLUSION

MCAs provide an opportunity to greatly accelerate applications. However, in order to harness the quickly growing compute resources of MCAs, applications must expose their thread-level parallelism to the hardware. We explore one common approach to doing this for large-scale multiprocessor systems: decomposing parallel sections of programs into many tasks, and letting a task scheduler dynamically assign tasks to threads.

Previous work has proposed software implementations of dynamic task schedulers, which we examine in the context of a key emerging application domain, RMS. We find that a significant number of RMS applications achieve poor parallel speedups using software dynamic task scheduling. This is because the overheads of the scheduler are large for some applications.

To enable good parallel scaling even for applications with very small tasks, we propose a hardware scheme to accelerate dynamic task scheduling. It consists of relatively simple hardware and is tolerant to growing on-die latencies; therefore, it is a good solution for scalable MCAs.

We compare the proposed hardware to optimized software task schedulers and to an idealized hardware task scheduler. For the RMS benchmarks we study, our hardware gives large performance benefits over the software schedulers, and it comes very close to the idealized hardware scheduler.

ACKNOWLEDGMENTS

We thank Trista Chen, Jatin Chhugani, Daehyun Kim, Victor Lee, Skip Macy, and Mikhail Smelyanskiy who provided the benchmarks. We thank Pradeep Dubey who encouraged us to look into this problem. We also thank the other members of Intel's Applications Research Lab for numerous discussions and feedback.

REFERENCES

[1] Pradeep Dubey, "Recognition, Mining and Synthesis Moves Computers to the Era of Tera," *Technology@Intel Magazine*, February 2005.

[2] Sanjeev Kumar, Christopher Hughes, Anthony Nguyen, "Carbon: Architectural Support for Fine-Grained Parallelism on Chip Multiprocessors," in *Proceedings of 34th International Symposium on Computer Architecture*, June 2007.

[3] *Intel® Thread Building Blocks Reference*, 2006. Version 1.3.

[4] *OpenMP Application Program Interface*, May 2005. Version 2.5.

AUTHORS' BIOGRAPHIES

Sanjeev Kumar is a Staff Researcher in the Corporate Technology Group. His research interests are parallel architectures, software, and workloads especially in the context of chip-multiprocessors. He received his Ph.D. degree from Princeton University. His e-mail is sanjeev.kumar at intel.com.

Christopher J. Hughes is a Staff Researcher in the Corporate Technology Group. His research interests are emerging workloads and computer architectures, with a current focus on parallel architectures and memory hierarchies. He received his Ph.D. degree from the University of Illinois at Urbana-Champaign. His e-mail is christopher.j.hughes at intel.com.

Anthony D. Nguyen is a Senior Research Scientist in the Corporate Technology Group. His research interests include developing emerging applications for architecture research and designing the next-generation chip-multiprocessor systems. He received his Ph.D. degree from the University of Illinois, Urbana-Champaign. Hise-mail is anthony.d.nguyen at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

Datacenter-on-Chip Architectures: Tera-scale Opportunities and Challenges

Ravi Iyer, Corporate Technology Group, Intel Corporation
Ramesh Illikkal, Corporate Technology Group, Intel Corporation
Li Zhao, Corporate Technology Group, Intel Corporation
Srihari Makineni, Corporate Technology Group, Intel Corporation
Don Newell, Corporate Technology Group, Intel Corporation
Jaideep Moses, Corporate Technology Group, Intel Corporation
Padma Apparao, Corporate Technology Group, Intel Corporation

Index words: chip multiprocessors, datacenters, tera-scale, QoS, cache, memory, platforms

ABSTRACT

We have entered an era of chip multiprocessor (CMP) platforms, where performance is delivered with the integration of more and more cores on a die. Tera-scale CMP architectures, consisting of several tens of physical cores and hundreds of hardware threads, are highly suitable for throughput computing especially in the server market place. In this paper, we start by highlighting tera-scale potential in datacenter environments. We show how a multi-tier datacenter workload that required tens (to hundreds) of platforms in the past can potentially map on to one (or a few) single-socket tera-scale CMP platforms running Virtual Machines (VMs) and thereby creating Datacenter-on-Chip (DoC) architectures.

Having introduced tera-scale DoC architectures, we then describe key challenges involved in providing high degrees of performance, scalability, and adaptability. Performance and scalability challenges point to the need for efficient handling of cache/memory/IO requirements when a large number of cores are actively running many workloads. Adaptability challenges highlight the need for dynamically allocating cache, memory, and I/O resources amongst the simultaneously running VMs in order to enable Quality of Service (QoS). To address scalability and adaptability challenges, we then propose and evaluate important tera-scale architectural features: (a) hierarchy of shared caches and large DRAM caches for better cache/memory scalability and performance, and (b) cache/memory QoS techniques to form Virtual Platform Architectures (VPAs). Based on a detailed evaluation, we show that these architectural features are highly beneficial for DoC tera-scale architectures.

INTRODUCTION

We have entered the era of CMP platforms with Intel's dual-core and quad-core processors [5, 8] flourishing in the mobile, desktop, and server marketplace. Within a decade, we expect to integrate more and more cores on-die and create tera-scale architectures consisting of several tens of physical cores and hundreds of hardware threads. Such tera-scale architectures are highly suitable for high-performance throughput computing especially in the server marketplace.

A decade ago, datacenters employed tens to hundreds of dual-processor and quad-processor server platforms (each running a single application) on an Ethernet fabric. However, recent trends show that most datacenters have started employing virtualization [21, 23, 31, 32] to consolidate multiple applications onto the same platform in order to improve efficiency, manageability, and overall cost [6]. With tera-scale architectures [7] comes the potential to accelerate the consolidation trend and potentially even enable small datacenters to run on a single (or a few) platforms, thus the term "Datacenter-on-Chip" (or DoC) architectures. In this paper, we use an e-commerce benchmark, TPC-W [29], to illustrate this by showing how an earlier configuration (with 60+ server platforms) can now potentially run on a single tera-scale DoC platform with 32 cores and 128 threads (4 threads per core).

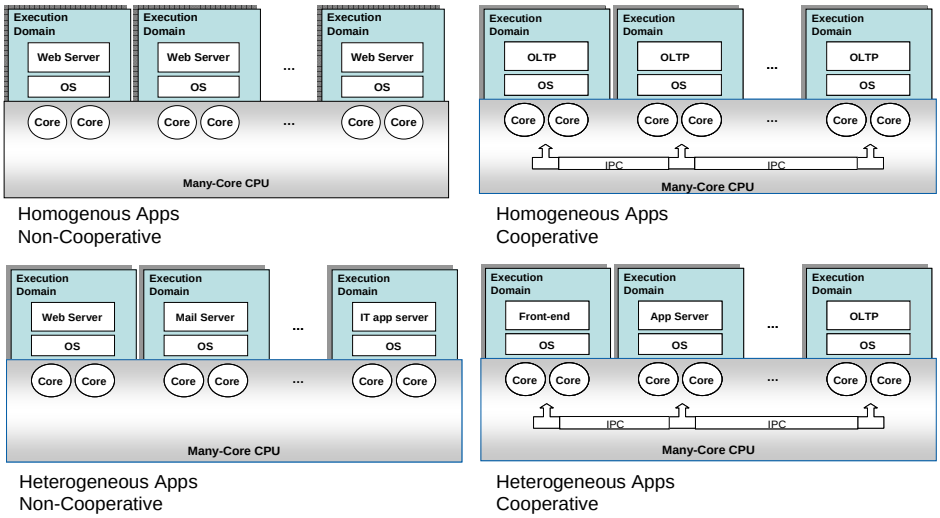


Figure 1: Datacenter-on-chip usage models: classification and examples

With tera-scale DoC architectures comes the challenge of designing a balanced platform with sufficient resources to sustain the large number of cores actively running VMs. In this paper, we evaluate the cache, memory, and I/O requirements as well as the behavior of the DoC architecture. We accomplish this by analyzing the TPC-W configuration as well as running detailed platform simulations that mimic multiple VMs running Online Transaction Processing (OLTP) workloads, Java* application server workloads, and even enterprise resource planning workloads simultaneously. To address the cache/memory scalability requirements, we show that (a) a hierarchy of shared caches is most suitable for DoC architectures since it maximizes performance and area efficiency when running virtualized server workloads and (b) integrating a large-capacity DRAM cache can significantly reduce the memory bandwidth requirements and thereby improve performance and scalability.

Another critical challenge in DoC architectures is that the performance of each VM can be highly non-deterministic since it depends heavily on the other VMs running simultaneously. Since an abundant number of cores is provided in tera-scale DoC architectures, the source of this non-determinism comes from interference in shared platform resources such as cache and memory. Through detailed simulations of simultaneously running VMs, we quantify the impact of this interference and the lack of QoS provided to each individual workload. Since datacenters typically provide service-level agreements, it is important to incorporate QoS hooks in the platform resources such as cache and memory. In this paper, we describe potential platform QoS mechanisms and evaluate the effectiveness of these mechanisms in improving the performance isolation provided to each VM.

DATACENTER-ON-CHIP USAGE MODELS AND TERA-SCALE ARCHITECTURES

In this section, we start by describing four classes of DoC usage models and then focus on one of them to highlight the potential of tera-scale architecture and describe the key challenges.

Datacenter-on-Chip Usage Models

Virtualization techniques make it possible to consolidate multiple server applications onto a single system. This usage model has been gaining momentum in enterprise datacenters because it improves resource sharing and usage, improves manageability, and reduces cost. We expect this trend to continue growing significantly in the coming years especially with the integration of more and more cores on the die. DoC essentially refers to the potential of multiple datacenter applications running simultaneously on a single-chip server platform. DoC usage scenarios can be classified into four broad categories based on (a) the types of applications being consolidated and (b) the level of communication and cooperation between the applications. Figure 1 illustrates the four DoC usage models that are explained further below.

- **Homogeneous/Non-Cooperating:** In this type of consolidation, multiple server applications of the same type are consolidated onto a single platform. However, these applications are independent in nature and no significant communication is required between the applications. A good example is the consolidation of a farm of Web servers that are serving Web pages and are load balanced. For the most part, these different

Web servers run on their own without having to communicate with each other.

- **Heterogeneous/Non-cooperating:** In this type of server consolidation, multiple different server applications are consolidated onto a single system. It is often the case in enterprise datacenters that servers are under utilized for a significant portion of the time when running a single application. The main motivation for this type of consolidation is to achieve maximum resource usage by consolidating multiple applications onto the same platform. In this type of consolidation, the applications being consolidated are still quite independent and do not need to communicate with each other. One example is that of consolidating an email server, file and print server, and user authentication server.
- **Homogeneous/Cooperating:** This type of consolidation occurs when a clustered application (e.g., database cluster) is consolidated onto a single system. Clusters use some sort of message passing either on a regular network fabric like Ethernet or on a more specialized fabric to communicate with each other. This communication turns into inter-VM communication once consolidated onto the same platform.
- **Heterogeneous/Cooperating:** This type of usage model occurs when multiple heterogeneous workloads that need to communicate with each other are consolidated. A good example of this type is where a multitiered application, like in TPC-W, is consolidated onto a single system. Here various tiers need to communicate with each other while servicing user requests. Hence inter-VM communication can be a significant factor, and handling this can be a challenge in virtualized environments, as we will see in the later part of this paper.

Mapping to Tera-scale Architectures

The DoC usage models described above can take advantage of more and more cores on-die since they have many applications (potentially multi-threaded) running simultaneously on a single platform. As a result, a tera-scale architecture with several tens of physical cores and hundreds of hardware threads integrated on the die is highly suitable for DoC usage. To illustrate the potential of tera-scale and highlight the challenges, we now focus on a case study of an e-commerce environment based on the TPC-W benchmark.

TPC-W [29] is a benchmark representative of an e-Commerce datacenter environment defined by the Transaction Processing Performance Council (TPC). The performance metric reported by TPC-W is the number of Web interactions processed per second (WIPS). Multiple Web interactions are used to simulate the activity of a retail store, and each interaction is subject to a response time constraint. The TPC-W benchmark is now obsolete; however, the e-Commerce workload that it represents is very relevant and important. A typical TPC-W setup contains several different application components (as shown in Figure 2):

- **Web Servers** process incoming HTTP requests and prepare responses to be sent to the clients.
- **Web Cache Servers** cache static and dynamic content for fast access to data.
- **Image Servers** serve static images that are part of the response Web pages.
- **Application Servers** provide the e-Commerce functionality and are responsible for processing customer orders and payments for goods, among other things.
- **Database Servers** hold inventory of product, description, availability, pricing and other information.
- **Load Balancer and other Infrastructure Servers** distribute processing load among different Web and image servers equally by directing incoming HTTP requests to the server with the least load.

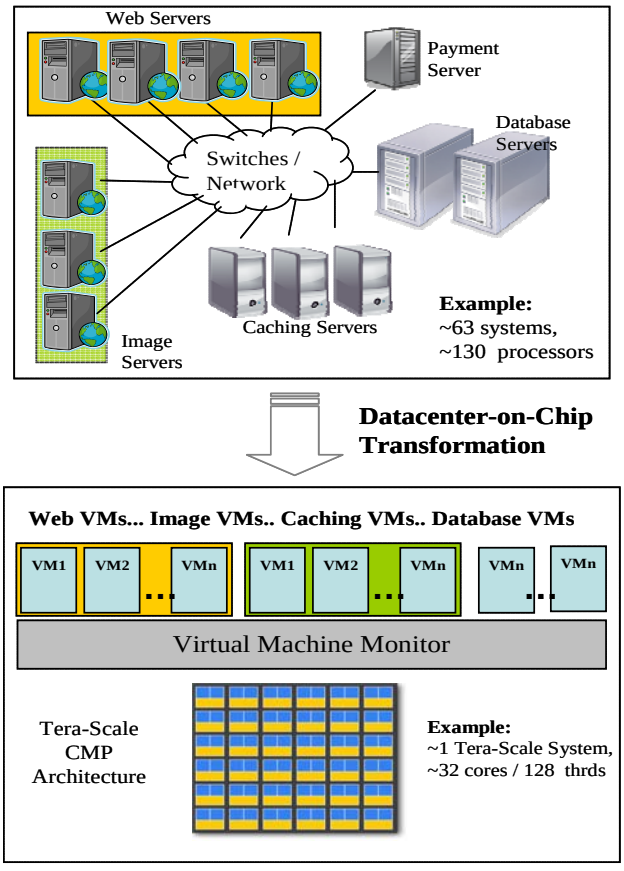


Figure 2: Mapping datacenter workloads to tera-scale

Table 1: Compute/cache/memory capacity data from TPC-W setup (example based on TPC-W publication [30])

Server Type	Oses	Num servers	Total Procs	Frequency (GHz)	Memory (MB)	Total Cache (I	CPU Util
Web server	Win 2K Server	26	52	1.26	19968	26624	80%
Web server/Other software	Win 2K Server	1	2	1.26	768	1024	80%
Image server	Win 2K Server	20	40	1.26	10240	20480	
Image server/Load Balancer	Win 2K Server	1	2	1.26	512	1024	
Database server	MS .NET EE	1	8	1.6	8192	8192	45%
Web cache	Win 2K Server	9	18	1.26	4608	9216	70%
Web cache	Win 2K Server	3	6	1.26	2304	3072	70%
Web cache	Volera	2	2	1.26	4096	1024	55%
Total		63	130		50688	70656	

Figure 2 shows how these different components are interconnected in a typical TPC-W setup of the past. TPC-W is a perfect example of understanding the requirements and behavior of consolidating multiple tiers (heterogeneous/cooperating) of a datacenter on a tera-scale CMP platform (bottom of Figure 2). Table 2 summarizes the number of systems, compute cores, cache and memory in an example configuration roughly based on a high-performing 2002 TPC-W publication [30]. As shown in the figure as well as the table, there are 63 systems employed in the example TPC-W configuration. Except for the database server, which employed four processors, all other systems consisted of two processors (without multi-threading). As a result, the total number of processors in the configuration was about 130. In a tera-scale CMP platform, we expect that a single processor socket could contain 32 cores each with 4 threads (SMT). As a result, the entire TPC-W example configuration can be potentially consolidated onto such a 32-core, 128-thread single-socket platform.

However, it is also critical that we take into account the amount of platform resources that are needed to support the execution of simultaneously running VMs of this nature. For example, Table 2 shows that the total cache capacity available in the TPC-W configuration adds up to 70MB in size. Given the area constraints and the fact that 32 cores will be integrated onto the die, our previous work [36] has shown that the amount of cache space available is likely to be less than 32MB. As a result, architectural techniques that enhance cache/memory scalability and performance need to be incorporated into the platform. We discuss these further in the next section.

Another key challenge in running heterogeneous VMs of this nature on the same platform is that they will contend for platform resources and interfere with each other. Given that these VMs are likely to get very different utility benefits from platform resources, and that the VMs are likely to be different in importance to the overall performance of the datacenter, it is important that we incorporate adaptability techniques in the platform so that resource usage can be dynamically controlled to provide performance isolation or QoS for DoC platforms. In the following section, we describe adaptability challenges and solutions to address these in tera-scale architectures.

Last but not least, it is also important to consider the overheads of virtualization on the DoC performance. In addition to the basic overhead of handling system calls, context switches, and interrupts for VMs, one primary concern in virtualized platforms is the overhead of I/O virtualization. For example, Figure 3 shows the overheads of virtualization for (a) transmitting network data to external platforms, (b) receiving network data from external platforms, and (c) inter-VM communication between VMs. The data shown in Figure 3 are based on measurements done on a recent Intel® Xeon® dual-core processor (3GHz) dual-socket platform [8] running the Xen hypervisor [3, 33]. The measurements show that (a) receiving network data at 1Gbps and processing requires 75% of CPU utilization under virtualization, (b) transmitting 1Gbps externally requires about 50% of CPU processing, and (c) communicating 1Gbps between VMs on the same platform requires about 70% of CPU utilization. Further, it should be noted that these compute cores are large out-of-order cores without multiple threads sharing the pipeline. As we design tera-scale processors, the use of smaller in-order cores with multiple threads sharing the pipeline may increase the associated processing overhead. However, since most of the cores in the example TPC-W configuration were underutilized (last column in Table 1), there is likely some headroom available to accommodate this extra I/O processing overhead. Extensions to techniques (such as Intel's I/O Acceleration Technology [14, 22]) are needed to minimize this overhead for a virtualized DoC environment. However, this is not covered in this paper.

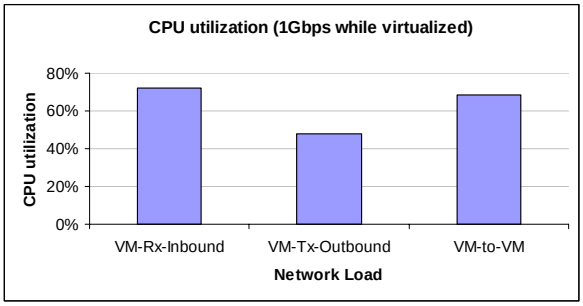


Figure 3: CPU overheads for network I/O virtualization

SCALABILITY CHALLENGES AND SOLUTIONS

As described in the previous section, the tera-scale architecture offers a high compute density (large number of cores and threads) that is attractive for DoC usage models. However, in order to provide high performance and scalability, it is important to carefully design a balanced platform with sufficient resources (cache, memory, I/O, etc.). In this section, we present the DoC scalability considerations and discuss potential solutions that address the key challenges.

The first challenge is that of providing sufficient cache space in order to reduce memory stalls and minimize memory bandwidth bottlenecks. Previous work [36] has shown that die area and cost will significantly restrict the amount of cache space that can be provided in tera-scale processors. In DoC usage models, the fact that several multi-threaded server applications will run simultaneously poses two potential considerations for cache hierarchy design: (a) since the threads within each server application tend to share code as well as data, cache space efficiency can be improved if these threads are allowed to share cache space, (b) since the cache space usage of each of the server applications can be quite different at different times in the execution, better utilization can be achieved if the cache space is shared. To take advantage of both of these sharing properties, we propose and evaluate a hierarchy of shared caches for tera-scale DoC platforms.

Figure 4 illustrates a three-level hierarchy of shared caches in a tera-scale platform. The hierarchy of shared caches starts an L1 (16K to 64K) that is private to the core but shared between the multiple threads within the core. The L2 (256K to 1M, mid-level) cache is also shared by multiple cores within a "node." The node forms the basic building block for the architecture. The L3 (8 to 32M, last-level) cache is logically shared by all of the nodes in the socket. However, since the L3 cache is quite large, it is physically distributed around the die in smaller "slices." A scalable interconnect connects all the L3 cache slices and the nodes. The benefits of sharing at each level is best explained with an example. Figure 5 compares the cache performance of private and shared L2 caches for an OLTP workload (based on the TPC-C [28]). As shown in the figure, a shared cache organization (e.g., 512K shared by four cores) is equivalent in cache performance to a private cache organization (four cores each with a 256K private L2 cache). This essentially shows a potential of 2X space efficiency with a shared cache organization. Similar benefits were found for other server workloads as well as for other levels of the hierarchy.

Having defined a cache hierarchy, the next major challenge is that of providing sufficient memory bandwidth to sustain the misses from the last-level cache. Figure 6 shows the cache scaling behavior of a consolidated server workload running on a last-level cache. These data were obtained from trace-driven simulations of four (8-threaded) workloads based on TPC-C [28], SPECjbb2005 [26], SPECjappserver2004 [25], and SAP SD/2T [24] running simultaneously on 32 single-threaded cores. The data show that consolidation workloads have good cache scaling behavior from 4MB all the way to 128MB of cache shared between the 32 cores.

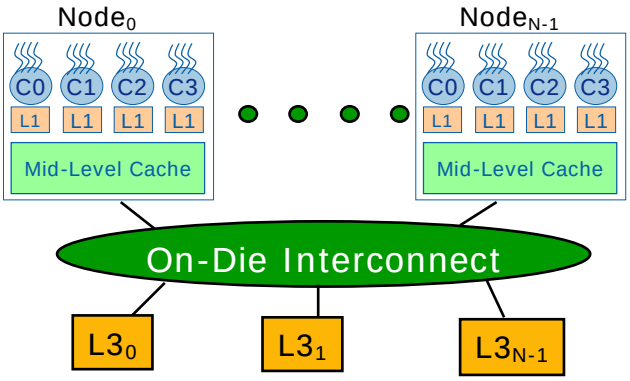


Figure 4: Tera-scale DoC hierarchy of shared caches

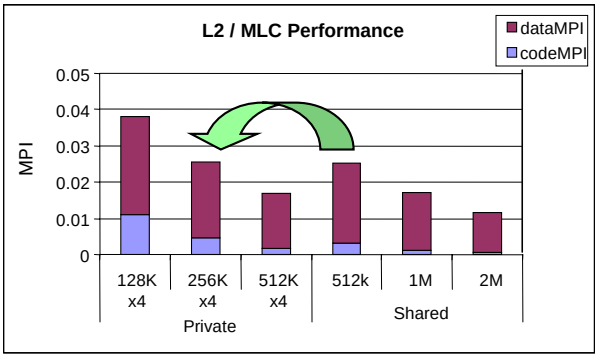


Figure 5: Tera-scale shared L2 cache benefits

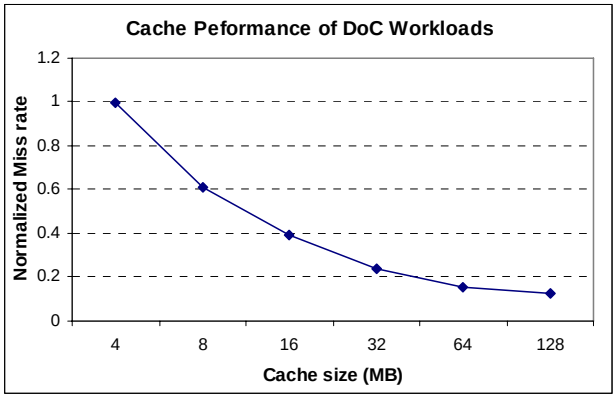


Figure 6: Tera-scale DoC L3 cache scaling behavior

To understand the memory bandwidth requirements of tera-scale DoC platforms, let us now consider the simulation configuration with 8MB L3 cache and 32 cores. In this configuration, we estimated that the bandwidth requirements can be as high as 20GB/s. Given that tera-scale processors may contain as many as 128 threads, the overall bandwidth requirements can be 100GB/s or higher. This in turn requires that a proportional number of memory channels be supported on the socket. Alternate solutions to solving the memory bandwidth bottleneck for tera-scale platforms could be the use of large capacity L4 caches. As shown in Figure 7, large capacity L4 caches can be implemented either as an additional package on the package (in a multi-chip package) or stacked (using 3D stacking technologies [1]). To understand the potential of large capacity L4 DRAM caches that can provide as much as twice the bandwidth at as little as one-third of the memory latency, we conducted simulations of a 32-core, 8MB L3 cache configuration with and without a 32MB or 64MB L4 cache. We found that significant performance benefits (from 10 to 40%) can be achieved depending on the organization of the DRAM cache, the exact bandwidth capability, and the latency benefits as compared to main memory latency. However, the key benefit is that of providing sufficient headroom in external memory so that the number of channels that is implemented can be reduced without affecting the performance.

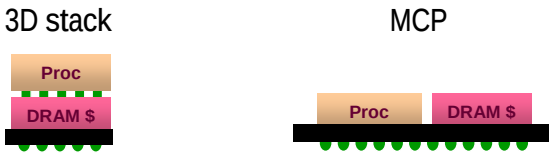


Figure 7: Tera-scale DoC L3 cache scaling behavior

Having addressed the cache/memory scalability challenges for tera-scale DoC architectures (using a hierarchy of shared caches and large L4 caches), we next turn our attention to adaptability concerns and solutions.

ADAPTABILITY CHALLENGES AND SOLUTIONS

Flexible and dynamic management of platform resources is important in DoC tera-scale architectures since multiple VMs will be running simultaneously. Traditionally, the execution environment (a virtual machine monitor (VMM) or hypervisor in DoC) attempts to control the visible resources (number of cores and memory capacity for instance). However, this alone will not suffice for CMP platforms where more cores might be available to run the virtualized applications simultaneously, but they end up contending for other (invisible) shared resources that have first-order performance impact [2, 4, 10, 16, 18, 34]. Key among these invisible resources are cache space and memory bandwidth. In addition to cache and memory, other resources that are shared include interconnects, micro-architectural resources in the core (shared between hardware threads), and power.

While sharing resources is generally the most efficient approach to maximize resource utilization, having no control over management of these resources can lead to loss of determinism, lack of performance isolation, and an overall lack of the notion of QoS provided to an individual application running on the platform. This has a very direct impact on the datacenter consolidation environments where more and more heterogeneous workloads are consolidated into a single platform contending for the shared hardware resources. Another important aspect to consider when managing shared resources is the relative importance of each of the consolidated applications. Not all applications consolidated may be of equal importance. The difference in priority could be based purely on the service level agreement provided to the customer or could be based on the relative throughput requirements of each of the consolidated applications. It could also be decided by the VMM layer based on the workload behavior (cache friendly, IOVM, etc.).

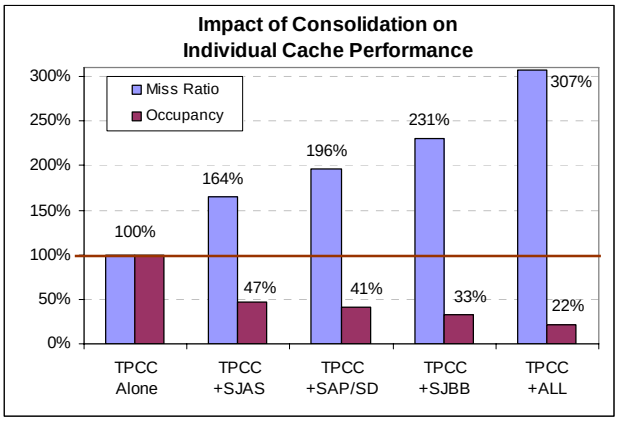


Figure 8: OLTP cache performance under consolidation

We start by studying the extent to which contention for shared cache space affects an individual OLTP application when running with one or more other consolidated applications. We performed a trace-driven simulation [9] of a 32-core processor with 8MB of last-level cache running (a) an 8-threaded OLTP application (based on TPC-C) running alone, (b) OLTP consolidated with an 8-threaded J2EE application server workload (based on SPECjappserver2004), (c) OLTP running consolidated with an 8-threaded ERP application (based on SAP SD/2T), (d) OLTP consolidated with a

8-threaded Java workload (based on SPECjbb2005), and (e) OLTP consolidated with all of the three above applications. Figure 8 shows the impact of consolidation on OLTP cache occupancy as well as OLTP miss rate. As the occupancy is reduced from 100% when running alone to as low as 20% when running with all other workloads, the miss rate goes up significantly (by as much as 3X). It should be noted that even though the compute resources available to the OLTP application remain the same when running alone and running in consolidated mode, the performance will be significantly affected due to the increase in miss rate.

QoS and Virtual Platform Architectures

Managing the allocation of shared resources in the platform is key to addressing the contention effects shown above and to providing performance differentiation, performance isolation, and the overall notion of QoS. Today, Intel and other processor manufacturers, support hardware virtualization features. While these features support functionally isolated VMs, they do not offer the ability to provide performance isolation. Our goal is to define mechanisms that allow VMs to transform into Virtual Platform Architectures (VPAs). A VPA is defined as a collection of virtual resources (i.e., some fraction of each of the physical shared resources) that a VM is provided. In this section we introduce our Platform QoS research that enables QoS-aware platforms and VPAs.

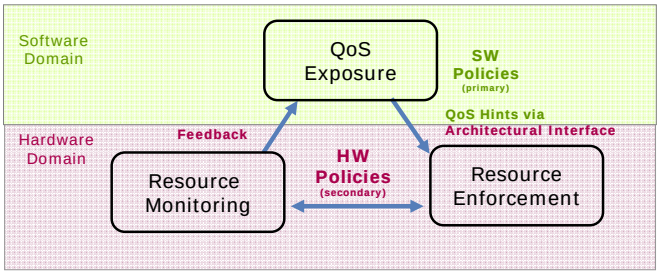


Figure 9: Platform QoS approach

Platform QoS

For DoC tera-scale architectures, there are three key questions that the Platform QoS research attempts to answer: (a) how much of each shared resource is an application or VM using (b) how can the resource allocation be modified to improve individual QoS or overall performance, and (c) what are the most appropriate interfaces and mechanisms needed between hardware and software to achieve QoS and VPAs?

The Platform QoS approach attempts to address these questions by enabling three major components: (a) monitoring, (b) enforcement, and (c) exposure. Figure 9 shows the components and their relationship in terms of information flow. The monitoring and enforcement components are implemented in hardware, whereas the QoS policies and exposure can either be guided by software or by hardware. The monitoring component keeps track of shared resource usage on a per-application or per-VM basis. The resource monitoring ability allows the platform to pass back information to the execution environment (VMM or hypervisor in DoC) to determine the VPA that each VM ends up with in a platform. In addition, providing this information back to the software domain allows the VMM to optimize scheduling decisions or pass down hints for resource enforcement. The monitoring ability may also be useful to the system administrator to determine (a) whether a VM should be migrated to a different platform (if it is getting too few resources consistently), (b) what QoS hints should be passed down to the platform to modify resource allocation, or (c) what the end-customer should be charged based on resource usage. Alternatively, the administrator may be able to set up a QoS policy that performs one or more of the above dynamically, based on monitoring data.

The resource enforcement component implements shared resource partitioning based on software guidance. This requires an architectural interface to be exposed to the execution environment that allows the specification of resource allocation requirements on a per-VM basis. While we expect QoS policies to be determined primarily by software, it is also important to allow a path for future platform optimizations that dynamically manage resources entirely in hardware. The resource enforcement component enables the VMM to create VPAs with a user-specified amount of resources. To achieve a scalable low-cost QoS solution, we propose resource partitioning and QoS exposure on a class of service basis instead of a per-VM basis. This is sufficient since it is unlikely that all of the VMs running on the platform need performance isolation simultaneously. Instead, one or more VMs can be mapped to a single class of service as specified by the VMM, and a smaller number of classes of service can be supported by the platform. In this paper, we use the terms “priority class,” “priority level” and “class of service” interchangeably.

To help clearly describe the Platform QoS approach and mechanisms required, we now present a case study using the shared cache as the platform resource.

QoS Case Study Using Shared Caches

Since contention to shared cache (e.g., last-level) is a key issue, we now describe the implementation considerations for shared cache monitoring, enforcement, and exposure (highlighted in Figure 10).

In the case of cache monitoring, the goal is to keep track of cache space consumed on a per-application or per-VM basis. In order to do so, the VMM needs to pass down a

unique identity (ID) to the platform for each running VM. This can be easily done by writing the ID to a new register, a Platform QoS Register (PQR), that is part of the processor architectural state. Since the ID is finite, it should be noted that the ID might have to be recycled amongst VMs (when the number of VMs is larger than the number of IDs). Once the ID is passed down, each load/store generated by the CPU is tagged with the ID so that it is passed down to the last-level cache. In the last-level cache, each cache line is tagged with the ID, and a global cache occupancy counter is also maintained per ID. When a line is evicted from the core, the appropriate cache occupancy counter is decremented. When a new line is allocated into the cache, the appropriate cache occupancy counter is incremented. The implementation can be optimized for area by employing set sampling techniques [37] if so desired.

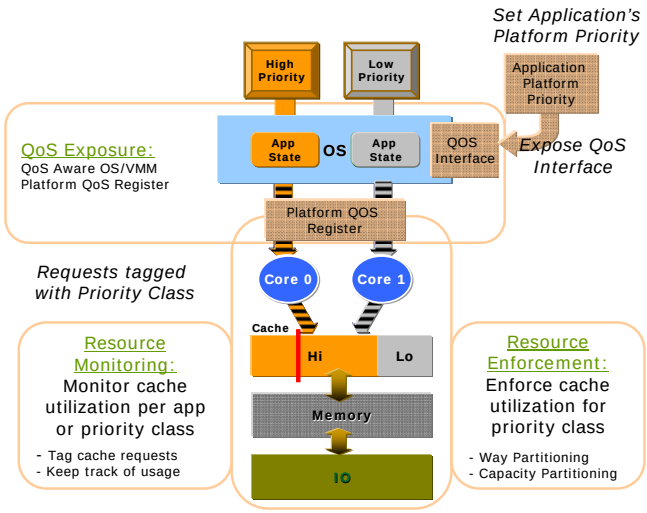


Figure 10: Cache QoS architecture and techniques

For cache enforcement, we are investigating the use of several forms of capacity partitioning. One potential approach attempts to limit the number of cache lines in the entire cache used by a certain class of service. Since the class of service is also associated with each cache line, this enforcement is accomplished by modifying the cache replacement policy to pick the next victim from a class that is currently exceeding its assigned cache quota.

For cache QoS exposure, we introduce the PQR. The PQR allows software to specify (a) the VM ID, (b) the class of service (also referred to as priority level or priority class) that this VM should be mapped to, and (c) an optional resource allocation target for that class of service. As described above, the VM ID is used by the platform to monitor cache occupancy per application. The class of service is used to guide the QoS-aware replacement decision.

To study the potential benefits of cache QoS enforcement we extended our trace-based cache simulations to implement cache enforcement. We conducted performance simulations of various consolidation scenarios where we limited the amount of cache space available to the low priority VM, but allowed high-priority VMs to allocate anywhere in the cache. In our example, we chose the OLTP application as the high-priority VM (with access to 100% of shared cache) and the three other consolidated applications as the low-priority VMs (limited to X% of the cache space). Figure 11 shows the OLTP miss rate as a function of X% (on the x-axis). As expected, reducing X from 100% to 12.5% improves the cache performance of the high-priority OLTP application significantly. It may be noted that this will negatively impact the performance of the low priority VMs, but that is expected as an outcome of performance differentiation and QoS.

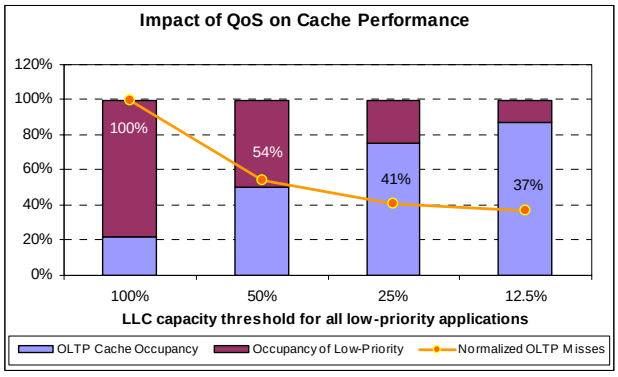


Figure 11: Case study of cache QoS benefits

In the previous sections, we focused on the cache-sharing impact and cache QoS implementations. However, the implications are similar for other shared platform resources. For example, memory bandwidth is another resource that has a direct impact on application performance. Memory QoS [11] can be achieved by implementing priority queues in the controller or enabling rate control of the request stream. Once all shared resources are enabled with QoS support, we could provide differentiated service to the individual VMs running on top of these resources. This combined with hardware-supported virtualization provides a complete VPA where functional and performance isolation is provided to VMs in a DoC architecture.

CONCLUSION

In this paper, we introduced DoC architectures and showed the potential of tera-scale platforms for DoC environments. The opportunity for more and more applications currently running on dedicated platforms to run on a tera-scale platform is tremendous, but it also

introduces some significant scalability and adaptability challenges that we need to address.

In this paper, we presented the scalability challenges for DoC in tera-scale platforms and described two important potential architectural features: (a) hierarchy of shared caches and (b) large-capacity L4 caches. We showed that enabling sharing at each level of the hierarchy can significantly maximize the space efficiency (e.g., sharing the mid-level L2 cache between multiple cores within a node provided a 2X better area efficiency as compared to private L2 caches). In addition, we also showed that large-capacity L4 caches (enabled either by 3D-stacking or a multi-chip package) can mitigate the memory bandwidth challenges for tera-scale platforms.

Last, but not least, we presented the adaptability challenges for DoC tera-scale environments. DoC environments suffer from the lack of performance isolation and performance differentiation since multiple simultaneously running VMs are contending for critical shared platform resources. We described our Platform QoS research that is investigating QoS techniques for resource monitoring and enforcement to enable performance isolation and differentiation. We showed how these QoS techniques allow us to transform VMs into VPAs. The end goal is to provide better QoS in tera-scale platforms for DoC environments.

Future work in this area is as follows. Research work along the lines of scalable cache/memory hierarchies [12, 27, 35, 19] and adaptable QoS techniques [4, 10, 11, 13, 15, 16, 17, 18, 20, 37] is a great start, but more and more emphasis on DoC usage models will be needed in the future.

REFERENCES

[1] B. Black et al., "Die Stacking (3D) Microarchitecture," *39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2006.

[2] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multiprocessor architecture," *11th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2005.

[3] "Xen Virtualization Technology," *Xen Source*, at http://www.xensource.com/xen/xen/*

[4] L. Hsu, S. Reinhardt, R. Iyer and S. Makineni, "Communist, Utilitarian, and Capitalist Policies on CMPs: Caches as a Shared Resource," *Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006.

[5] Intel Corporation, "Intel Dual-Core Processors," at http://www.intel.com/technology/computing/dual-core/.

[6] Intel Corporation, "Multiply Virtualization Maximize Server Harmony," at http://www.intel.com/business/technologies/virtualization.htm?iid=servproc+marquee_virtualization

[7] Intel Corporation, "Tera-scale Computing," at http://techresearch.intel.com/articles/Tera-Scale/1421.htm

[8] Intel Corporation, "Intel® Xeon® Processor 5000 Sequence," at http://www.intel.com/products/processor/xeon5000/index.htm?iid=servproc+body_xeon5000

[9] R. Iyer, "On Modeling and Analyzing Cache Performance using CASPER," *Int'l Symposium on Modeling, Analysis and Simulation of Computer & Telecom Systems,* Oct. 2003.

[10] R. Iyer, "CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms," *18th Annual International Conference on Supercomputing (ICS'04),* July 2004.

[11] R. Iyer, L. Zhao, et al., "QoS Policies and Architecture for Cache/Memory in CMP Platforms," *SIGMETRICS*, 2007.

[12] C. Kim, D. Burger, S. W. Keckler, "Nonuniform Cache Architectures for Wire-Delay Dominated On-Chip Caches," *IEEE Micro 23(6),* pp. 99–107, 2003.

[13] S. Kim, D. Chandra, and Y. Solihin, "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," *13th Int'l Conf. on Parallel Arch. & Complication Techniques (PACT),* Sept. 2004.

[14] K. Lauritzen, T. Sawicki, et al., "Intel® I/O Acceleration Technology Improves Network Performance, Reliability and Efficiently," *Technology@Intel Magazine,* at http://www.intel.com/technology/magazine/communications/intel-ioat-0305.htm

[15] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs," *10th IEEE Symposium on High-Performance Computer Architecture*, Feb. 2004.

[16] K. Nesbit, et al., "Fair Queuing Memory Systems," in *Proceedings. of Annual International Symposium on Microarchitecture (MICRO)*, June 2006.

[17] K. Nesbit, et al., "Virtual Private Caches," *International Symposium on Computer Architecture (ISCA)*, June 2007.

[18] M. K. Qureshi and Y. N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," *Int'l Symposium on Microarchitecture (MICRO)*, June 2006.

[19] M. Qureshi, A. Jaleel, et al., "Adaptive Insertion Policies for High Performance Caching," *International Symposium on Computer Architecture (ISCA)*, June 2007.

[20] N. Rafique, W.T. Lim and M. Thottethodi, "Architectural Support for Operating System-Driven CMP Cache Management," *Int'l Conference on Parallel Architectures and Compilation Technology (PACT 2006)*, Sept. 2006.

[21] P. Ranganathan and N. Jouppi, "Enterprise IT Trends and Implications on Architecture Research," *11th Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2005.

[22] G. Regnier, S. Makineni, R. Illikkal, R. Iyer, et al., "TCP Onloading for Datacenter Servers," *IEEE Computer*, 2004.

[23] M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," *IEEE Transactions on Computers*, 2005.

[24] Sap America Inc., "SAP Standard Benchmarks," at http://www.sap.com/solutions/benchmark/index.epx*

[25] SPECjAppServer2004, at http://www.spec.org/jAppServer/*

[26] SPECjbb2005, at http://www.spec.org/jbb2005/*

[27] E. Speight, H. Shafi, L. Zhang, R. Rajamony, "Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors," *32nd International Symposium on Computer Architecture (ISCA)*, June 2005.

[28] TPC-C Benchmark, at www.tpc.org/tpcc/

[29] TPC-W Benchmark, at www.tpc.org/tpcw/*

[30] TPC-W Publication, at http://www.tpc.org/results/FDR/tpcw/ibm.x440.w.fdr.02091201.pdf*

[31] VMware Corporation, Server Consolidation and Containment with VMware Virtual Infrastructure, at http://www.vmware.com/pdf/server_consolidation.pdf*

[32] R. Uhlig, et al., "Intel Virtualization Technology," *IEEE Transactions on Computers*, 2005.

[33] T. Deshane, D. Dimatos, et al., "Performance Isolation of a Misbehaving Virtual Machine with Xen, VMware and Solaris," at http://people.clarkson.edu/~jnm/publications/isolationOfMisbehavingVMs.pdf

[34] T. Y. Yeh and G. Reinman, "Fast and Fair: Data-stream Quality of Service," *Int'l Conf. of Compilers, Architecture and System For Embedded Systems (CASES)*, July 2004.

[35] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors, *32nd International Symposium on Computer Architecture (ISCA-32)*, Madison, 2005.

[36] L. Zhao, R. Iyer, et al., "Performance, Area and Bandwidth Implications on Large-Scale CMP Cache Design," *Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, Feb. 2007.

[37] L. Zhao, R. Iyer, et al., "CacheScouts: Fine-Grain Monitoring of Shared Caches in CMP Platforms," to appear in *16th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2007.

AUTHORS' BIOGRAPHIES

Ravi Iyer is a Principal Engineer with the Systems Technology Lab in Intel's Corporate Technology Group. His current research focus is on large-scale CMP architectures and technologies. Before joining STL, he held positions in the Communications Technology Lab (working on IO acceleration research) and in the Enterprise Products Group (working on server architecture and performance). He received his Ph.D. degree in Computer Science from Texas A&M University. He has filed 20+ patent applications and published 70+ papers in the areas of computer architecture, server design, network protocols/acceleration, workload characterization, and performance evaluation. He has held program committee member positions in various conferences and workshops (HPCA 2006, PACT 2007, IISWC 2007, etc.). He is also an Associate Editor for *IEEE Transactions on Parallel and Distributed Systems* (IEEE TPDS) and is currently guest co-editor for a special issue on CMP architectures. His e-mail is ravishankar.iyer at intel.com.

Ramesh Illikkal is a Senior Researcher in the Systems Technology Lab at Intel. His research interests are in CMP, server architectures, virtualization, and memory

hierarchies. He received his Masters degree in Electronics from Cochin University of Science and Technology. His e-mail is ramesh.g.illikkal at intel.com.

Li Zhao received her Ph.D. degree in Computer Science from the University of California, Riverside. She is currently a Senior Engineer in the Systems Technology Laboratory at Intel. Her research interests include computer architecture, network computing, and performance evaluation. She is a member of the IEEE. Her e-mail is li.zhao at intel.com.

Srihari Makineni is a Senior Researcher in the Systems Technology Lab at Intel. He has been working at Intel for more than 11 years. His research interests include cache/memory subsystems, interconnects, networking, and large-scale CMP architectures. Makineni received an M.S. degree in Electrical and Computer Engineering from Lamar University, Texas. His e-mail is srihari.makineni at intel.com.

Don Newell is a Senior Principal Engineer in the Systems Technology Lab at Intel. His research interests include server architecture, networking, and I/O acceleration. Newell received a B.S. degree in Computer Science from the University of Oregon. His e-mail is donald.newell at intel.com.

Jaideep Moses is a Senior Engineer in the Systems Technology Lab at Intel. Prior to this, Jaideep worked in the Communication Technology Lab on I/O acceleration research. He also worked in the former Enterprise Products Group focusing on modeling, simulation, and analysis of platform architecture and design including simulation-based verification of a coherence protocol. His current research focus is on large-scale CMP platform architecture analysis and design. Jaideep received his M.S. degree in Computer Science from the University of Texas at El Paso. His e-mail is jaideep.moses at intel.com.

Padma Apparao is a Senior Researcher in the Systems Technology Lab at Intel. Padma received her Ph.D. degree from the University of Florida and has been working on performance analysis of server workloads. Her current research interest is in the areas of virtualization and large-scale CMP architecture analysis. Her e-mail is padmashree.k.apparao at intel.com.

Note: The use of SPEC or TPC benchmark configurations and traces in this paper is purely for analysis and illustration. They are not intended to provide any indication of performance of a specific platform.

THIS PAGE INTENTIONALLY LEFT BLANK

Media Mining—Emerging Tera-scale Computing Applications

Yurong Chen, Corporation Technology Group, Intel Corporation
Eric Li, Corporation Technology Group, Intel Corporation
Wenlong Li, Corporation Technology Group, Intel Corporation
Tao Wang, Corporation Technology Group, Intel Corporation
Jianguo Li, Corporation Technology Group, Intel Corporation
Xiaofeng Tong, Corporation Technology Group, Intel Corporation
Patricia Wang, Corporation Technology Group, Intel Corporation
Wei Hu, Corporation Technology Group, Intel Corporation
Yimin Zhang, Corporation Technology Group, Intel Corporation
Yen-Kuang Chen, Corporation Technology Group, Intel Corporation

Index words: tera-scale computing, media mining, video processing, parallel computing, performance analysis

ABSTRACT

With the exponential increase in media data on personal computers and the Internet, it is critical for end users to efficiently manage metadata to find the information they are looking for. Media mining refers to a technique whereby a user can retrieve, organize, and manage media data. However, most media-mining applications are compute intensive, and they require tera-operations per second. This paper focuses on how tera-scale computing enables new usage models with media-mining techniques. Several representative media-mining usage examples are explored in detail.

First, we look at how these new usage models are enabled by a different kind of parallelism. For maximum performance, we provide a general parallel framework to abstract various parallelisms. We also present a detailed architectural performance analysis of several representative workloads on a dual-socket, quad-core system and on a 32-core Chip Multiprocessor (CMP) simulator. The results indicate that these media-mining applications have no obvious limits on concurrency and are amenable to future large-scale, multi-core architectures. They can take full advantage of tera-scale computing power in the form of thread-level parallelism to meet users' needs.

Because the underlying techniques and fundamental algorithms in media mining are widely used in other applications, many of our findings are applicable to other emerging applications as well.

INTRODUCTION

Rapid advances in the hardware technology of media capture, storage, and computation power have contributed to an amazing growth in digital media content. As content generation and dissemination grows, extracting meaningful knowledge from large amounts of multimedia data becomes increasingly important. Media mining is a kind of technology that helps end users search, browse, and manage large amounts of multimedia data [1]. It yields a wide range of emerging applications with various mass-market segments, e.g., image/video retrieval, video summarization, scene understanding, visual surveillance, digital home entertainment, smart health care, etc. Most of these applications are very complicated and have real-time or even super-real-time processing demands, which require tera-scale computing power to make them usable.

In this paper, we present several media-mining applications that require target architectures capable of delivering tera-scale computing. Our study shows that today's single-core processor system performance is 10x–1000x slower for acceptable human interactions. To accelerate these compute-intensive applications, we exploit the inherent data and function parallelism of these workloads. Our experiments show that with proper parallelization, these workloads can scale well, achieving a speedup of up to 7.5x on a 2-socket, quad-core machine and a speedup of up to 30x on a 32-core CMP simulator.

This paper is organized as follows. First, we explore several media-mining usage models and their key techniques. Next, we present several different parallel schemes and a general parallel video-mining framework. Then, we show our performance analysis results of the parallelized workloads.

MEDIA-MINING APPLICATIONS

Media mining has a huge number of emerging applications with different usage models. We highlight three typical usage models developed at Intel.

Media-Mining Usage Models

- **Sports video analysis**: Broadcast sports videos are very popular on television. Using highlights detection, consumers can quickly retrieve specific video clips without having to browse through the whole video. Sports video analytics can be viewed from the perspective of an editor. Based on a predefined semantic intention, an editor combines certain multimedia content elements and their temporal layout to achieve the desired highlighted events. Hence, detecting highlighted events is similar to a reverse process of authoring. The system framework consists of three levels: low-level audio/visual feature extraction, mid-level semantic keywords generation, and high-level event detection [8]. To minimize the semantic gap between low-level features and high-level events, we use mid-level semantic "keywords" followed by a classifier to infer events of interest. Our sports video analysis system can work with a multitude of sports including soccer, hockey, badminton, tennis, and diving. Given a video in a specific domain with predefined semantic intentions, the system can extract the desired events and features and interpret a summarization output video in terms of high-level semantics.

- **Personal video editing**: Home videos are increasingly popular as digital video cameras become more user friendly and portable. However, because home videos for the most part are shot by amateurs, shaking, blurring, under-exposure artifacts, and redundant content are always present. Therefore, the demand for an automated home video editing system [2] is high. Such a system has to be able to recognize how many people and how many scenes are involved, mine the relationship between various people and scenes, and synthesize a short artistic video clip from a long raw video. A typical personal video editing system includes three key modules: intelligent analysis, adaptive selection, and seamless composition. The first module extracts the multi-modal and multi-level audio-visual features; the second module selects the most interesting, important, and informative content; and the third module produces a near-professional story with incidental music. The overall automated home video editing system must be easily extended to the personal video recorder and digital home entertainment system.

- **Personal video retrieval**: A personal video retrieval system is a desktop application that works much like the Google desktop search to help end users manage more and more personal multimedia data from all kinds of mobility digital camera devices. In response to a user query, the personal video retrieval application finds the relevant video clips from a large video database such as from movies, TV, sports games, and home videos. Generally, a retrieval system first extracts low-level audio/visual features from videos, and then detects semantic concepts (keywords) to represent the video content. Finally, a query engine returns retrieval results based on the user's query and on a similarity model. The query can be text keywords, image examples, hand-drawn sketches, or short video clips, and the output is relevant video clips ranked not only by their content similarity to the query, but also by their importance, according to a concept-link relationship analysis. To gradually improve system performance during the query procedure, the system provides user-friendly relevant feedback and active learning modules.

Key Media-Mining Techniques

Although the above usage models are quite different from one another, the underlying technologies are common and can be extended to a broad range of media-mining applications. In this paper, four key techniques are extracted from previous usage models to show how media-mining applications are built.

- **Sports keyword detection:** The mid-level module generates semantic "keywords" from the previously described low-level extraction. Listed below are some keywords in sports video analysis. These keywords are used as input for high-level event detection.

  - View type: Based on color histograms of each frame, we can obtain the dominant color to segment the playing field region. We then classify each frame as a global view, medium view, close-up view, and out of view [5].

  - Play-field: A Hough transform from digital image processing is used to detect field boundaries and penalty box sections. Then a decision-tree-based classifier determines the play

position according to the slope and position of the lines.

- Replay: In broadcast sports videos, to capture clues for significant events, there typically is a replay following an important event. At the beginning and end of each replay, there is generally a logo flying in high speed. We detect logos to identify replays by discovering repeat video segments through dynamic programming [6].
- Audio keywords: There are two types of audio keywords: commentator's excited speech and referee's whistle: these have a strong correlation to key events in the game such as a foul, a goal, or player entanglements. A Gauss Mixture Model (GMM) is used to detect keywords from low-level audio features including Mel frequency Cepstral coefficients (MFCC), energy, and pitch [7].

- **Human detection and tracking:** Human detection and tracking is a significant and challenging task in many application scenarios. Different from rigid objects, humans are articulated and jointed by several human-parts, which may lead to pose variance, self-occlusion, etc. In human detection, the first problem is to select the proper features to characterize human regions/parts: Haar wavelets [3] and orientation histograms are mostly used to do this. The second problem with human detection is to use a discriminator to determine whether there are humans and where they are if they are present. The Boosting learning-based detector is preferred [3]. It is an aggressive learning algorithm that produces a strong classifier by choosing features in a family of simple classifiers and combining them linearly. Then a cascaded structure is introduced in order to quickly reject the background regions. Human tracking is essentially finding body regions or parts that correspond with successive frames by using data association and occlusion inference techniques.
- **Face detection and tracking:** Face detection and face tracking have been an important technology and pre-requirement for many person-analysis relevant applications, such as face recognition/identification, emotion analysis, and cast indexing. Face detection has been studied for many years. Viola and Jones Boosting learning-based detection algorithms are the most successful algorithm to date [2]. Recently, some improvements are proposed to enable the algorithm to handle multi-view faces more efficiently for high-quality videos [12]. Generally, Boosting-based face detection characterizes image regions by very simple Haar wavelet features, and it learns cascade detection from a training set to separate a face set from a non-face set. In the detection phase, the learned detector will slide by a window over the image to detect whether the window contains a face or not. Face tracking [13] is an extension of face detection technology, which can detect a person's continuous faces from a video sequence. Spatial and temporal constraints are employed to avoid much unnecessary calculation. Since it detects faces only in predicted face image regions, it doesn't waste time scanning all the positions of every frame.
- **Concept ontology indexing:** Concept ontology indexing represents multimedia data by large-scale concept ontology for indexing and fast retrieval. There are several concept lexicons for multimedia: large-scale ontology for multimedia (LSCOM) [9] is the most popular. LSCOM currently contains about 1000 concepts that are relevant to objects, people, locations, scenes, and events. LSCOM has been successfully used by the TREC video retrieval evaluation (TRECVID) hosted by NIST [10]. Concepts are detected from more than 20 low-level MPEG-7 compatible audio/visual features, e.g., color histogram, Gabor texture, shape context, edge histogram, motion, and MFCC audio features, etc. Given these low-level features, a supervised classifier (such as an SVM) is learnt for each concept from a training set to identify whether the concept exists or not in each video shot [11]. Employing all of the concept detectors, a video shot is therefore represented and indexed by the semantic concept ontology that makes next-stage search similar to text retrieval.

Common Characteristics in Media Mining

Three attributes of media-mining applications can be summarized as follows:

- First, a media-mining system is basically a bottom-up framework as shown in Figure 1. The framework is a three-layer architecture, i.e., low-level feature extraction, mid-level semantic keywords detection, and high-level concept detection. In processing, low-level visual/audio/textual features are extracted from raw media data. Then in the second layer, mid-level features or keyword concepts are detected from low-level features to bridge the semantic gap between low-level features and high-level concepts. Finally, high-level modules infer the desired concepts in the semantic keyword spaces.

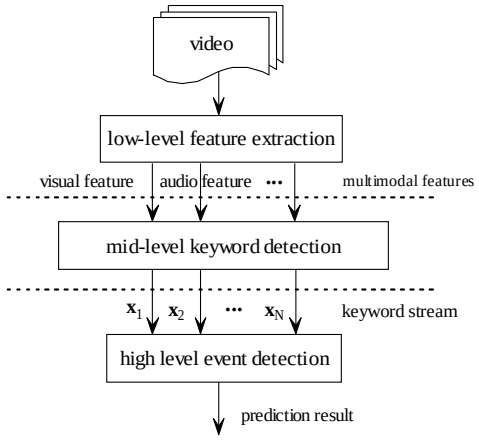


Figure 1: General video-mining framework

- Second, media mining is a hybrid technique of computer vision, pattern recognition, machine learning, and data mining. For example, human detection/tracking techniques involve Haar and HoG feature extraction from video frames, Boosting (cascade learning) training-based candidate detection, and associate rule learning from quite large examples to identify relationships between articulations. In these techniques, Haar and HoG features are essentially computer vision methods; Boosting is a famous machine-learning algorithm; and associate rule learning is a typical data-mining method.
- Third, media-mining applications usually combine multiple components. For example, in the automatic home video editing application, the application needs to recognize people, mine the relationship between people, and synthesize a short artistic video clip from a long raw video.

Media mining has mass-market potential and is therefore quite a suitable and important proxy not only for workload analysis on future architectures, but also for developing parallel programming models for multimedia applications. Furthermore, due to its similar framework for different usage models, we only use one technique as an example to study its computational requirements.

Computational Requirement: a Case Study

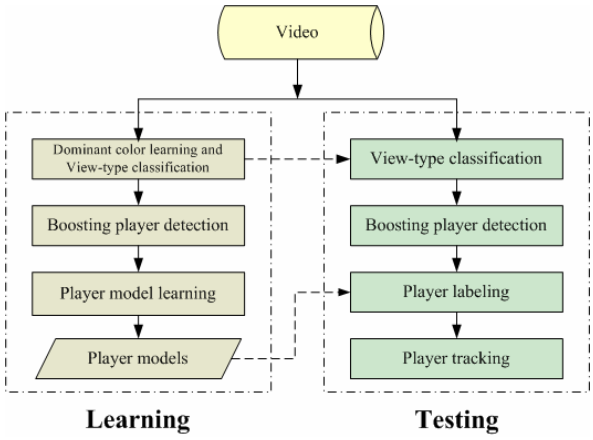


Figure 2: Flowchart of player detection, tracking and classification

In the sports domain, we look at multiple player detection, tracking, and classification in broadcast soccer video for our example. Its flowchart is shown in Figure 2 [4]. To make the algorithm robust and adaptive, we construct the background (playfield) color model and three player appearance models (Team A, Team B, and Referee) through unsupervised learning procedures. In the learning phase, the background color model is obtained by accumulating color histograms over hundreds of frames in the video in HSV color space. Player appearance models are learned by player sample collection with a boosted player detector, color histogram representation, and clustering. In the testing phase, we first perform background segmentation, playfield extraction, and view-type classification. Only global views are selected for player detection. We then apply a boosted cascade of Haar features for player detection on each foreground pixel within the playfield. Multiple detections will usually occur around each player after scanning the image. We merge adjacent detected rectangles and get final detections with proper scale and position. In the player classification procedure, each player sample is represented by the learned codebook histogram. We calculate the Bhattacharyya distance between the histogram and each sub-model. The player sample is assigned the sub-model's label by the nearest neighbor rule. With this procedure, players are labeled as Team A, Team B, Referee, or Outlier (if the minimum distance is larger than a threshold). Player tracking is performed by efficient forward and backward nearest neighbor data association. We take both binary mask overlap and color histogram intersections in player upper-body as observations within a certain spatial displacement range to find the optimal player regions correspondence, and we generate players' trajectories across frames.

Figure 3 is an example of player tracking results, in which white ellipses and rectangles indicate two teams' players and a black rectangle is the referee.

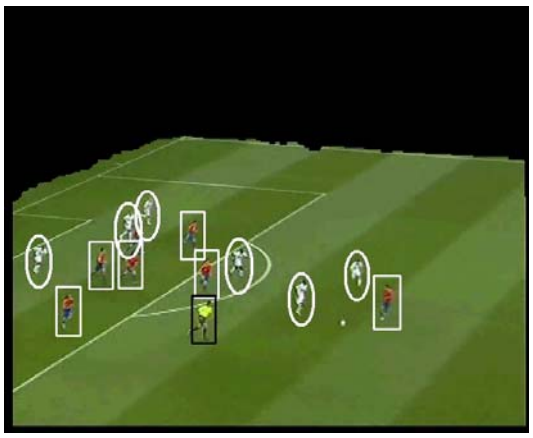


Figure 3: Player tracking results on soccer video

Player detection is achieved by background elimination and a boosted cascade of Haar features. In this paper, we only show the detailed detection procedure since this procedure is most compute intensive compared to tracking and classification. The cascade detector with multiple stages has the capability of quickly rejecting the regions and focus on the harder-to-classify windows. The number of features selected in each stage is different depending on the expected performance and sampling criterion. Therefore, increasingly complex classifiers are combined sequentially. This improves both the detection speed and efficiency.

- Input: image frame, background model
- Playfield elimination and view-type classification
- Player detection
  For each scale
    Scan each point to be detected
    For each point
      Evaluate its response with cascaded stages
      Calculate normalized constant
      For each stage
        Evaluate the response
        For each selected Haar feature
          Calculate Haar feature response
          Normalize Haar feature response
          Get weak classifier response
        Accumulate all Haar response
      If verified by the threshold, begin next stage;
      else, label the point as negative, break;
    If pass all stages, label the point as positive
- Post-processing to merge adjacent detection instances
- Output: vector of player regions (rectangles)

Based on the above description, one can easily infer its computation complexity, which is proportional to the size of the video frame, the number of weak classifiers, and the number of scales. For player tracking between two adjacent frames, it is proportional to the number of players and player size. For player classification, it is linear to the number of players, player size, size of codebook, and size of sub-model. For an MPEG-2 video, the frame size is 720x576; we use about 1000 weak classifiers and three different scales. Thus, one minute of MPEG-2 video will need 1.86 tera-operations. Its serial processing speed on today's processors is about 3 frames per second, which is 10x slower than real-time.

MEDIA-MINING PARALLEL FRAMEWORK

With the boom in multi-core processors and the prevalence of shared memory processing, it is important to exploit thread-level parallelism within applications to take advantage of next-generation microprocessor capability. In this section, we present the parallelization methodology, characterize different parallel schemes, and provide insights for parallelizing these media-mining applications on future multi/many-core systems. Besides the parallelization study, we also made intensive optimizations, e.g., genetic loop-level optimization, SIMD acceleration, and cache-conscious optimizations, to provide a fully optimized baseline for further workload parallelization and analysis.

Video-Mining Parallelism

Most video-mining applications can be partitioned into three modules: video decoding, feature extraction, and post processing. We use an MPEG-2 video decoder to divide the input video stream into a number of consecutive decoded frames. Then we use a feature extraction module to extract a set of visual features from these decoded frames. This process continues until all the frames are processed. Finally, all the feature results are fed into a post-process module to detect the final visual information. The breakdown of execution time indicates that the video decoding and image processing modules are the most time consuming. The post-processing module is extremely fast and is therefore not the focus of this paper.

We use a top-down analysis methodology to analyze the coarse-grained parallelism in each module and the whole application. In general, people tend to use data-domain decomposition rather than functional-domain decomposition to take advantage of the inherent parallelism in multimedia applications. Though fine-grained parallelism within each module is of interest, we don't explore this kind of parallelism as it's not profitable because of serial regions and insufficient parallelism in these modules. Therefore, we choose

coarse-grained parallelism to explore both functional domain decomposition and data domain decomposition within each task with the goal of load balancing, and we examine the issue of scalability when using a large number of processors.

A task-level parallel scheme uses the producer-consumer model, where the video decoder works as a producer, generating a sequence of video frames, while the image processing modules act as a consumer, operating on decoded frames to obtain the corresponding visual feature information for each frame. This multi-threading scheme is very similar to the task queue model provided by the Intel® OpenMP extension [14], which provides an efficient way to exploit functional-domain decomposition. The video decoder serves as the task producer to encapsulate the decoded frame as a task and conceptually put it in the task queue, and all the other worker threads will wait until the task is available. Though this parallel scheme is straightforward, it may experience limited scalability performance on a large number of processors when the ratio between feature extraction modules and video decoders is not sufficiently high.

A static data slicing parallel scheme slices the raw data into several video bit-stream chunks. Each thread performs a similar routine to that of a sequential application: decoding the bit-stream chunk and extracting features from the decoded frames. Because the raw video stream is split manually, each thread has to find the new sequence synchronization position. Therefore, there is an explicit synchronization between two adjacent threads to guarantee no excess or loss of decoded frames. In addition, the static data-slicing scheme may experience a load imbalance problem when the work is not evenly distributed across threads.

To take advantage of both task-level and data-level parallel schemes, **a dynamic hybrid parallelization approach** is proposed to combine these two schemes. At first, we decompose the video stream into several chunks to exploit the data-domain decomposition, and then we exploit the functional-domain decomposition on each particular chunk of data as previously mentioned. In this parallel scheme, there are multiple queues to buffer tasks. Master threads are responsible for video decoding and for enqueueing the feature extraction tasks in different queues. Worker threads fetch tasks from their associated queue and execute the tasks. Further, in order to reduce load imbalance, we use the work stealing strategy. When one queue is empty, it will steal tasks from other non-empty queues and execute the tasks on the idle physical processor. With work stealing enabled, the load imbalance disappears. In addition, due to reduced contention on the access to each queue, the synchronization overhead is reduced significantly.

Figure 4 illustrates the hybrid task-stealing scheme. The whole video is partitioned into four chunks and assigned to four thread groups. Within each thread group, a task queue is implemented with one master thread and three worker threads. The task will not be migrated to other queues unless its private task queue is out of tasks.

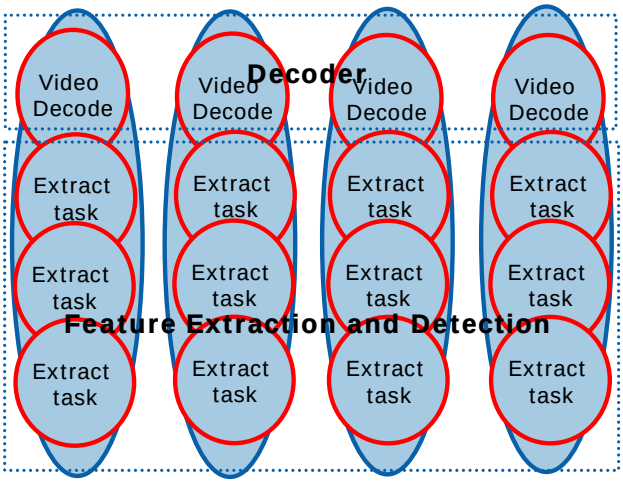


Figure 4: Dynamic hybrid task-stealing scheme

In summary, the dynamic hybrid parallelization scheme has several advantages. First, it significantly improves performance by manipulating multiple queues of video data in parallel, which reduces the competition for shared resources. Second, it solves the load imbalance issue by enabling dynamic task scheduling and stealing. Finally, the hybrid scheme provides enough flexibility by specifying the number of decoding and worker threads to maximize system resources utilization and deliver good scalability. However, from the perspective of programming, this dynamic hybrid parallelization approach is the most difficult of the three parallel schemes to build.

Parallel Pattern in Video Mining Applications

Because of the difficulty of parallelization in these media-mining applications, we construct a universal parallel video-mining framework to encapsulate the parallel scheme and provide an ease-of-use interface to the programmer.

The video-mining parallel framework [15] is built in C++, and OpenMP is the default parallel language. It includes the parallel implementation, an abstract interface, and a set of configuration parameters. There are four primary components in this framework:

- An image-processing engine that serves as the interface to invoke the user codes in the library and

perform feature extraction functionalities for each decoded frame.

- A video-decoding engine that acts as an interface to enable most video codec standards with parallel support.
- A portal video-mining function that serves as an interface to link the user codes with the framework.
- Configuration parameters and core image data structures.

The parallel video-mining framework has several advantages. It provides a unified parallel computing environment for video-mining applications. Programs written in this framework can be automatically parallelized and efficiently executed on a multi-core architecture. The run-time library takes care of the details. Furthermore, this framework is easily extensible and maintainable. The programmer can extend it to meet new requirements.

To summarize, video-mining workloads have abundant parallelism. The dynamic hybrid parallelization approach that combines both functional-domain decomposition and data-domain decomposition can achieve optimal parallel performance. In addition, the particular execution pattern of video-mining applications can be abstracted into a parallel video-processing framework to help programmers easily construct a parallelized video-mining application.

PERFORMANCE ANALYSIS ON MULTI-CORE SYSTEMS

In this section we analyze three typical media-mining workloads (Player, Face and Shot detection), which are parallelized via our video-mining parallel framework. To generate best-performing executable codes, the Intel 9.1 OpenMP compiler tool chain and highly optimized OpenCV and IPP library [16] are used. Furthermore, we also use the Intel VTune™ Performance Analyzer [17] to identify the hotspots in functional profiling and guide the optimizations. To characterize the parallel performance, the Intel® Thread Profiler is used to quantify the parallel performance metrics, i.e., synchronization, locks, load imbalance, etc.

We evaluate the scaling performance of these parallel media-mining workloads on a real multi-core machine and a large-scale CMP simulator. The multi-core platform is a dual-socket, quad-core machine, with two Intel® Core™2 Quad processors running at 2.33GHz. Each socket has four cores, and each core is equipped with a 32KB L1 data cache and a 32KB L1 instruction cache. The two cores on one chip share a 4MB L2 unified cache. The maximum Front-Side-Bus (FSB) bandwidth is 21GB/s. In addition to the existing multi-core system, we further study these media-mining applications' performance on a large-scale CMP simulator with cycle-accurate simulation to see how they will scale with the increasing number of cores. We assume a very high main memory bandwidth so that we do not artificially limit the scalability of the modules.

For the workloads studied in this experiment, we choose application parameters and datasets so as to represent realistic executions. For Player detection, we used a 30-minute MPEG-2 soccer video as the input. For Face and Shot detection, we used a 10-minute MPEG-2 movie video as the input.

Performance Scalability Analysis

Our video-mining workloads scale very well as the number of threads increases, as shown in Figures 5 and 7. That is, media-mining applications can efficiently use the computational power provided by multi-core processors.

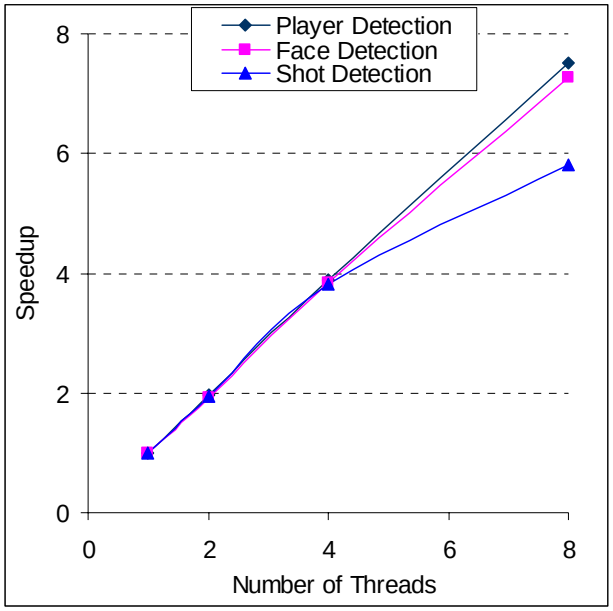


Figure 5: Scalability of parallel video-mining workloads on an 8-core system

However, as also shown in Figure 5, our workloads, in particular, Shot detection, do not have linear scaling on the 8-core system. To fully understand the scaling-limiting factors on an 8-core system, we characterize the parallel performance from the perspective of the high-level parallelization overhead, e.g., synchronization penalties, load imbalance, and sequential regions, and from the detailed memory behavior, e.g., cache miss rates and FSB bandwidth.

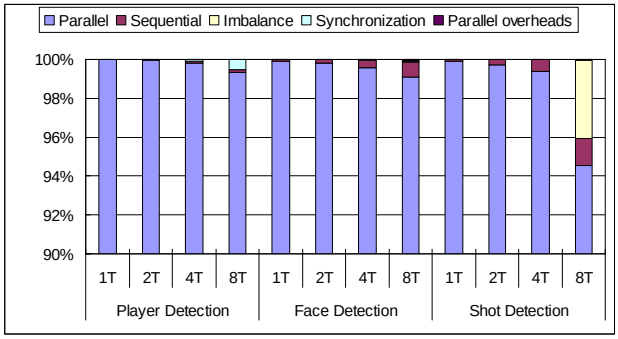


Figure 6: Execution time breakdown

In general, our parallelized workloads expose good parallel performance metrics. Figure 6 depicts the parallel profiling metrics for these three workloads. The higher the parallel region, the better speedup can be achieved on highly threaded architectures. Shot detection has slightly more load imbalance than other workloads. Because of frame dependency, it is more challenging to implement two-level task queues in Shot detection than in other workloads. In Shot detection, we use the static scheduling scheme, which leads to a slightly higher load imbalance. Nonetheless, the profiling information suggests these parallel video-mining workloads expose good parallel performance metrics. If we assume the parallel region can scale perfectly, the three workloads should achieve the theoretical speedups of 7.95, 7.93, and 7.56, respectively, on eight cores. They are higher than the results shown in Figure 5. Therefore, we believe the scalability of our workloads is limited by some other factors that are discussed in the next subsection.

On the simulated 32-core CMP system with a huge amount of memory bandwidth, two selected parallel video-mining workloads have very good scalability, as depicted in Figure 7. First, the size of the serial sections in the applications is reasonably small—the serial code accounts for much less than 1% of the execution time for the one-thread runs. Second, there is little contention on the locks: the locking overhead does not increase with the thread number due to coarse-grained parallelism. Third, the load imbalance is not a major issue; most of our video-mining workloads adopt a dynamic hybrid parallelization scheme. In short, when we assume a very high main memory bandwidth so that we do not artificially limit the scalability of the workloads, these applications scale very well.

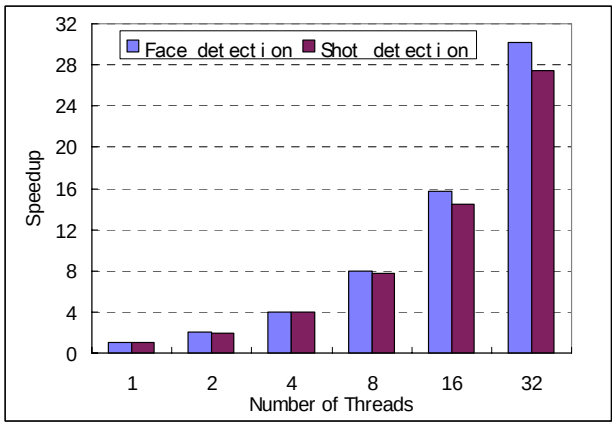


Figure 7: Scalability of two video-mining workloads on a 32-core CMP simulator

Memory Behavior Analysis

Besides the general parallel performance metrics, the memory subsystem also plays an important role in scalability. As shown earlier in Figure 6, our workloads with good parallel performance metrics should achieve the theoretical speedup of 7.6–7.9x on 8 cores, if the parallel region can scale perfectly. We now investigate why these workloads cannot achieve this perfect scaling performance from the perspective of the memory subsystem. We use the Intel VTune Performance Analyzer and a command-line tool for hardware-based performance counter sampling to further analyze the memory behavior of the applications on the real system, e.g., system memory bandwidth and L1/L2 cache miss rates.

Our first observation is that average bus bandwidth is not limiting the scalability of these workloads on the 8-core system. Figure 8 shows how the average FSB bandwidth utilization varies with the number of threads. The bandwidth usages of all workloads are far below the 21GB/s capacity supported by the system. This seems to indicate bus bandwidth does not limit the scalability of our workloads on the 8-core system.

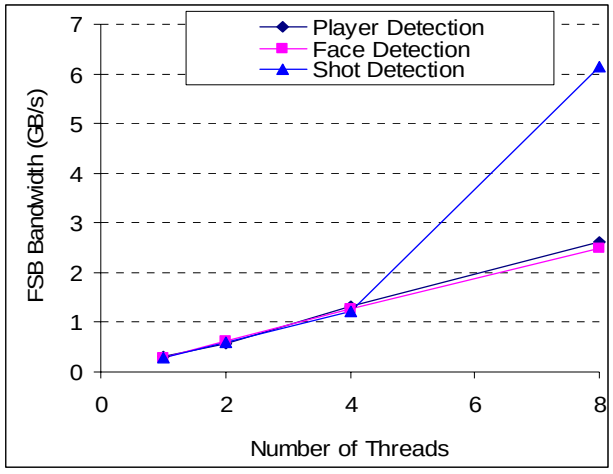


Figure 8: Average FSB bandwidth utilization vs. number of cores

Although workloads are not bounded by the average bandwidth usage, the scalability is limited by the instantaneous bandwidth usage. We perform interval sampling of the memory subsystem behavior over time. Figure 9 shows a representative phase of the bandwidth usage over time for the single-threaded Shot detection workload on a single core. It goes without saying that there are some bursty memory access behaviors—the instantaneous bandwidth usage is much higher than the average bandwidth usage. In particular, one of the modules demands about 7x more bandwidth over the average bandwidth. When the bandwidth demand of the module is higher than the system's capability, its speedup from 8 cores is less than 3x, and it becomes the bottleneck of scalability. In short, the workload is not able to scale perfectly when the instantaneous bandwidth usage is higher than the system's capability. This is what limits the scalability.

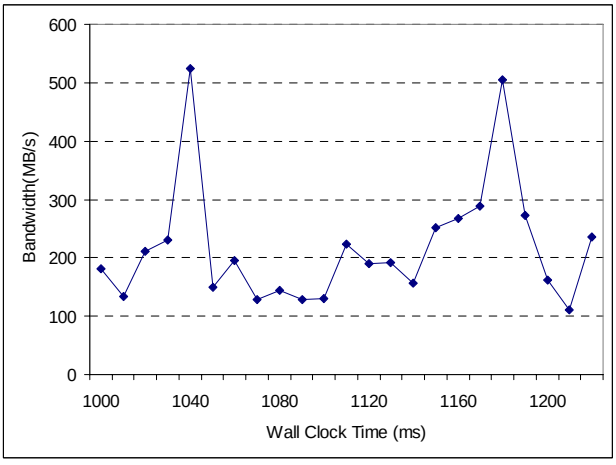


Figure 9: Bandwidth usage over time for single-threaded Shot detection workload

Additionally, there is a significant increase in bandwidth usage from four threads to eight threads for Shot detection. Figure 10 shows that L1 cache miss rates vary little with the number of threads, while L2 cache performance deteriorates when scaling the thread count. In particular, the external memory access rate for Shot detection increases from 0.05 bytes per instruction for a single thread to 0.30 bytes per instruction for eight threads. Because we exploit coarse-grained parallelism for these three workloads, each thread operates on a large private working set, about 32MB per thread for Player detection, 8MB per thread for Face detection, and 4MB per thread for Shot detection. As the total working set size increases with the number of threads, there are more L2 cache misses for more threads. For Shot detection, while the working set of four threads fits well into 16MB L2 caches, the working set of eight threads cannot fit. This explains the significant increase in cache misses from four threads to eight threads. Together with the instantaneously high bandwidth usage, the speedup of Shot detection from four threads to eight threads is much slower than the speedup from two threads to four threads.

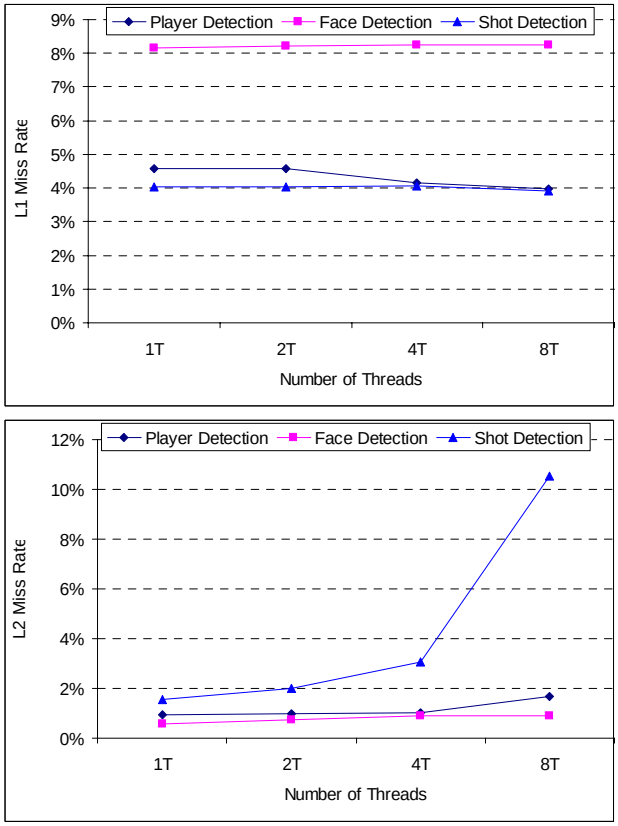


Figure 10: L1/L2 cache miss rates

To summarize, most of the video-mining workloads demonstrate fairly good parallel performance on both existing multi-core systems and future large-scale CMP platforms. As most of them can be partitioned into a large

number of parallel tasks, they have little lock overhead and serial region. Since the workloads are parallelized in coarse-grain fashion, which exposes a huge working set with the increase in thread numbers, large cache size and sufficient memory bandwidth will be necessary to enable large-scale, video-mining computing. To reduce the working set sizes and the external bandwidth usage in the future, we may need to exploit fine-grain parallelism. This could be a tradeoff between memory subsystem performance and parallelism overheads.

CONCLUSION

Media mining can help us retrieve, organize, and manage the exponentially growing media data easily. We explored several usage models in media mining and showed that most applications require tera-scale computing. To efficiently use the processing power provided by multi-core processors, we studied common parallelization schemes and proposed a general parallel framework for these media-mining applications. Furthermore, we conducted a performance analysis of several representative media-mining workloads on an 8-core system and a 32-core CMP simulator. Our analysis shows they have no obvious limits on parallelism. With a proper parallelization scheme, future large-scale CMP systems can deliver real-time performance for these media applications. Taking advantage of next-generation tera-scale computing platforms, new usage models in media mining will be enabled.

ACKNOWLEDGMENTS

We acknowledge the encouragement and help that we have received from Dr. Bob Liang, Director of the Applications Research Lab. Our thanks also go to Prof. Haizhou Ai, Prof. Jianming Li, Prof. Zhijian Ou, Prof. Lifeng Sun, Prof. Shiqiang Yang, and Prof. Bo Zhang from Tsinghua University for collaborating with us on the media-mining project and providing some original source code for our analysis. We also thank the reviewers for their valuable comments.

REFERENCES

[1] Chabane Djeraba, *Multimedia Mining: A Highway to Intelligent Multimedia Documents*, Kluwer, Norwell, 2002.

[2] X.S. Hua, L. Lu, and H.J. Zhang, "AVE - automated home video editing," *ACM Multimedia*, 2003.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *IEEE Int'l Conf. on Computer Vision and Pattern Recognition* (CVPR), 2001.

[4] J. Liu, X. Tong, W. Li, T. Wang, and Y. Zhang, "Automatic player detection, labeling and tracking in broadcast soccer video," accepted by *BMVC* 2007.

[5] A. Ekin, A.M. Tekalp, and R. Mehrotr, "Automatic soccer video analysis and summarization," *IEEE Trans. on Image Processing*, 12(7), July 2003, pp. 796–807.

[6] H. Bai, W. Hu, T. Wang, X. Tong, and Y. Zhang, "A novel sports video logo detector based on motion analysis," *International Conference on Neural Information Processing* (ICONIP), 2006.

[7] M. Xu, N. Maddage, C. Xu, M. Kankanhalli, and Q.Tian, "Creating audio keywords for event detection in soccer video," *IEEE International Conference on Multimedia & Expo* (ICME), 2003.

[8] J. Li, T. Wang, W. Hu, M. Sun, and Y. Zhang, "Two-dependence Bayesian network for soccer highlight detection," *IEEE International Conference on Multimedia & Expo* (ICME), 2006.

[9] L. Kennedy, "LSCOM lexicon definitions and annotations (version 1.0)," *DTO Challenge Workshop on Large Scale Concept Ontology for Multimedia, Columbia University ADVENT Technical Report* #217-2006-3, March 2006.

[10] NIST, TREC Video Retrieval Evaluation, at http://www-nlpir.nist.gov/projects/trecvid/*

[11] J. Cao, Y. Lan et al., "Intelligent multimedia group of Tsinghua University at TRECVID 2006," in *Proceedings TRECVID*, 2006.

[12] C. Huang, H. Ai, et al., "Vector boosting for rotation invariant multi-view face detection," *IEEE International Conference on Computer Vision* (ICCV), 2005.

[13] Y. Li, H. Ai, C. Huang, and S.H. Lao, "Robust head tracking with particles based on multiple cues fusion," *HCI/ECCV 2006, LNCS 3979*, pp.29–39.

[14]E. Su, X. Tian, M. Girkar, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," *4th European workshop on OpenMP* (EWOMP), 2002.

[15]E. Li, W. Li, C. Dulong et al., "User transparent parallel video mining library," *PMUP workshop*, 2006.

[16]Intel® Integrated Performance Primitives, at http://www.intel.com/software/products/IPP

[17] Intel® VTune™ Performance Analyzer, at http://www.intel.com/software/products/VTune

AUTHORS' BIOGRAPHIES

Yurong Chen is a researcher at the Microprocessor Technology Lab, Beijing. Currently, he conducts research on parallel processing of emerging applications, scalable workloads, and benchmarking and performance analysis for next-generation microprocessors/platforms. He joined Intel in 2004. Before that he did two years' postdoctoral research on large-scale scientific computing in the Institute of Software, Chinese Academy of Sciences. He received his Ph.D. degree from Tsinghua University in 2002. His e-mail is yurong.chen at intel.com.

Eric Li is a researcher in the Microprocessor Technology Lab, Beijing. Currently, he is working on media-mining technology development and performance analysis on multi-core architecture. Prior to this, he was involved in several projects related to bioinformatics, multimedia, and parallel computing. He received his M.S. degree from Tsinghua University in 2002 and joined Intel that same year. His e-mail is eric.q.li at intel.com.

Wenlong Li is a researcher in the Microprocessor Technology Lab, Beijing. Currently he is working on algorithmic and workload analysis on data-mining applications. Before this, he did research in loop compilation techniques for IPF architecture. He received his Ph.D. degree from Tsinghua University in 2005 and joined Intel that same year. His e-mail is wenlong.li at intel.com.

Tao Wang is a researcher in the Microprocessor Technology Lab, Beijing. Currently, he conducts research on video-mining, computer-vision, and machine-learning techniques. At Intel, he has been involved in several projects related to visual tracking, bioinformatics, soccer highlights detection, cast indexing, video summarization, and concept detection, etc. He received his Ph.D. degree in Computer Science from Tsinghua University in 2003 and joined Intel the same year. His e-mail is tao.wang at intel.com.

Jianguo Li is a researcher in the Microprocessor Technology Lab, Beijing. Currently, he works on multimedia mining and parallel algorithm design and implementation. He has been involved in several projects related to sports video analysis and content-based media mining/retrieval. He received his Ph.D. degree from Tsinghua University in June 2006 and joined Intel after graduation. His e-mail is jianguo.li at intel.com.

Xiaofeng Tong is a researcher in the Microprocessor Technology Lab, Beijing. His work is on personal desktop multimedia applications. His technical interests include computer vision and pattern recognition. He received his Ph.D. degree in Computer Science from the Institute of Automation, Chinese Academy of Sciences in 2006. His e-mail is xiaofeng.tong at intel.com.

Patricia P. Wang is a researcher in the Microprocessor Technology Lab, Beijing. Her current research focuses on video mining, machine learning, and pattern recognition. She joined Intel in July 2006. She received her Ph.D. and B.S. degrees from the Department of Computer Science and Technology, Tsinghua University, China, in 2006 and 2001, respectively. Her e-mail is patricia.p.wang at intel.com.

Wei Hu is a researcher in the Microprocessor Technology Lab, Beijing. His research interests include many areas of artificial intelligence, computer vision, and data mining. He is currently working on video-mining projects, such as commercial detection in TV programs and automatic speaker recognition in videos. Before joining Intel, he was a Research Associate for the Department of EEE at the University of Hong Kong (1998-1999). He received his Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences in 1998. His e-mail is wei.hu at intel.com.

Yimin Zhang is a researcher in the Microprocessor Technology Lab, Beijing. He leads a team of researchers working on various statistical computing techniques and their scalability analysis, recently focusing on media mining, data mining, etc. He joined Intel in 2000. At Intel, he has been involved in several projects related to natural language processing and speech recognition, especially focusing on Chinese-named entity extraction and DBN-based speech recognition. He received his B.A. degree from Fudan University in 1993, his M.S. degree from Shanghai Maritime University in 1996, and his Ph.D. degree from Shanghai Jiao Tong University in 1999, all in Computer Science. His e-mail is yimin.zhang at intel.com.

Yen-Kuang Chen is a Principal Engineer in the Microprocessor Technology Lab at Santa Clara, California. His research interests include developing innovative multimedia applications, studying the performance bottleneck in current architectures, and designing next-generation microprocessors/platforms. He has 10+ US patents, 25+ pending patent applications, and 75+ technical publications. He is one of the key contributors to Supplemental Streaming SIMD Extension 3 in Intel Core 2 processor family. He received his Ph.D. degree from Princeton University. His e-mail is yen-kuang.chen at intel.com.

High-Performance Physical Simulations on Next-Generation Architecture with Many Cores

Yen-Kuang Chen, Corporate Technology Group, Intel Corporation
Jatin Chhugani, Corporate Technology Group, Intel Corporation
Christopher J. Hughes, Corporate Technology Group, Intel Corporation
Daehyun Kim, Corporate Technology Group, Intel Corporation
Sanjeev Kumar, Corporate Technology Group, Intel Corporation
Victor Lee, Corporate Technology Group, Intel Corporation
Albert Lin, Corporate Technology Group, Intel Corporation
Anthony D. Nguyen, Corporate Technology Group, Intel Corporation
Eftychios Sifakis, Corporate Technology Group, Intel Corporation
Mikhail Smelyanskiy, Corporate Technology Group, Intel Corporation

Index words: Physical simulations, chip multiprocessor, many cores, parallel scalability, memory bandwidth

ABSTRACT

Physical simulation applications model and simulate complex natural phenomena. The computational complexity of real-time physical simulations far exceeds the capabilities of modern unicore microprocessors, which are limited to only tens of billions floating-point operations per second (FLOPS). However, the advent of multi-core architectures promises to soon make processors with trillions of FLOPS available. Such processors are also known as tera-scale processors. Physical simulations can exploit this huge increase in computational capability to increase realism, enable interactivity, and enrich a user's visual experience.

In this work, we study physical simulation applications in two broad categories: production physics and game physics. After parallelization, the benchmark applications achieve parallel scalabilities of 30×–60× on a simulated chip-multiprocessor with 64 cores.

We examine the memory requirements of physical simulation applications and find that they require cache sizes in excess of 128MB and main memory bandwidths in excess of hundreds of GB/s for real-time performance. A radical re-design of the memory hierarchy may be necessary for the multi-core tera-scale era to provide good scaling for this type of application.

INTRODUCTION

The booming computer games and visual effects industries continue to drive the graphics community's seemingly insatiable desire for increased realism, believability, and speed. In the past decade, physical simulation has become a key to achieving the realism expected by audiences of games and movies. Physical simulation models the laws of physics to simulate life-like movement and interaction among objects, such as rigid and deformable bodies, human faces, cloth, and water.

Physical simulation can be used in a variety of settings such as weather prediction, movie special effects, and computer games. Complex natural phenomena such as ocean waves crashing on a shore, a flag waving in the wind, or bricks falling from a collapsing tower are modeled by means of numerical simulation of physical laws. Modeling different natural phenomena requires a diverse set of techniques, algorithms, and data structures, making physical simulation both complex and general. Computation and memory requirements are extremely demanding. This makes the workloads a challenging target for current as well as future architectures.

In this paper, we examine applications involving physical simulation for production environments and for gaming. For production physical simulation, we study the PhysBAM package from Stanford University [5, 11], which is used by several special-effects and film production companies, including Pixar and Industrial

Light and Magic. The goal is to recreate the visual experience of a human observing a natural phenomenon. For gaming physical simulation, we study the open source ODE package [13]. This package provides similar functionality to the widely used commercial Havok Effect package from Havok. The goal of physical simulation in gaming is to make real-time interactions between objects as accurate as possible. The difference in goals for the two physical simulation domains leads to different choices for algorithms and data structures. However, these two domains do have many similar characteristics.

One common characteristic of production and gaming physical simulation is a need for significant acceleration. On a 4-way Intel® Xeon® processor 3.0GHz system, with 16GB of DDR2-3200 and three levels of cache on each processor (16KB L1, 1MB L2, and 8MB L3), the production physics workloads take 5 to 188 seconds to process a single frame. These workloads have hundreds of thousands to a few million entities (tetrahedra/grid cells) interacting with each other. In contrast, for game physics workloads, only a thousand objects can currently interact in real time. Acceleration by an order of magnitude or more will allow improved accuracy, modeling of new effects, and even interactive or real-time production applications. Multi-core processors are now common, and we expect the number of cores to increase steadily for the foreseeable future, so that multi-core processors capable of executing applications tens of times faster than today's processors are on the horizon. Such processors would improve the speed and realism of production-quality or real-time game physical simulation applications. However, for an application to harness the computational power of such a multi-core processor, it must effectively utilize multiple threads. Parallelization of a large code base as used by production or game physics applications is not trivial, especially when the target parallel scalability is tens of threads.

Another similarity in requirements for the two categories of physical simulation applications is high-bandwidth requirements. The size of the data scales with increasing resolution or number of objects in the simulation. Input sizes are often millions of volume elements or tens of thousands of objects. This leads to memory footprints that are tens of megabytes (i.e., larger than typical caches). These applications therefore require either much larger caches or a large main memory bandwidth.

Our contributions are as follows:

- We have parallelized six state-of-the-art physical simulation applications (fluid dynamics [4], human face simulation [12], and cloth simulation [2] for production physics and convex body collision [1, 3], game cloth [7], and game fluids [9] for game physics). In parallelizing these workloads, we employed various techniques which include parallelizing loops/graph operations and using alternative algorithms for better scalability.
- We simulated and analyzed the scalability of these applications using cycle-accurate simulation of a chip-multiprocessor with 64 cores. The workloads studied achieved a parallel scaling of 30× to 60× for 64 cores.
- We perform a detailed analysis of the memory requirements of these applications. Our study finds that future physical simulation workloads demand cache sizes close to 100 megabytes or physical main memory bandwidths in the hundreds of GB/s.

PHYSICS SIMULATION PIPELINE

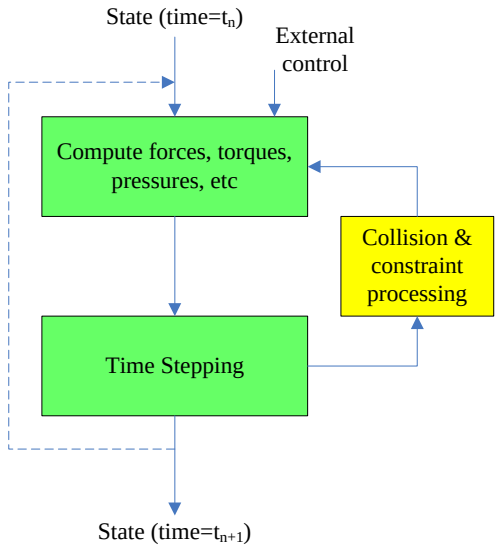


Figure 1: Overview of physical simulation

Figure 1 shows a typical time step in a physical simulation application. For each time step of a simulation, a physical simulation application takes as input the state of the simulated scene (e.g., positions, orientations, and velocities of all objects), as well as external control information (e.g., what the player is doing in a game). The application then computes the physical processes that potentially lead to an updated state (e.g., force, torque, or pressure generation). Depending on the scheme used, this information will be used to advance the state forward in time (e.g., time integration of the laws of motion), yielding a new candidate state. Should phenomena such as collisions or other constraints be triggered, the state may be updated in response to this collision or constraint, and the force computation/time stepping phases will be repeated, possibly with a smaller time step.

While game and production physics share the same iterative process, they exhibit important differences. These differences stem from the execution time requirements of their domains. *Production physics* is primarily used for special effects in movies and other off-line simulations. The execution time limit for these environments is typically a few minutes per frame in order to simulate the complete effect in a reasonable amount of time (e.g., less than a day). *Game physics*, on the other hand, is concerned with real-time simulation used in computer games. Thus, the execution time limit is at most tens of milliseconds per frame. Both areas of physical simulation have the goal of providing maximum visual plausibility within their execution time requirements.

We now describe how we model some specific phenomena.

Fluid Simulation

Production physics: Simulated water volumes are key elements in an increasing number of feature films, making fluid simulation (a.k.a., Computational Fluid Dynamics, or CFD) very common in the special effects industry today. Our production-quality fluid simulation application models a body of water with a free surface (as opposed to water flowing in a pipe or other airtight container). The application uses a combination of a three-dimensional grid and a set of particles [4]. The simulation tracks the velocity and pressure of the water in each grid cell. It computes how velocity and pressure change at each time step using incompressible Navier-Stokes equations. This is very computationally expensive, and it becomes much more so as the number of grid cells goes up. Unfortunately, unless prohibitively large grid resolutions are to be used, the grid cannot accurately represent intricate geometrical features of the water surface (such as thin sheets and droplets). Therefore, particles are sprinkled around the surface and advected along with the fluid. The updated positions and velocities of these particles are used to enhance the resolution of the water surface.

Game Physics: While CFD is the method of choice for high-fidelity simulation of fluids, its high computational requirements necessitate off-line rendering. Game physics therefore uses much faster, although less accurate, techniques. Smoothed Particle Hydrodynamics (SPH) has recently emerged as a popular technique for interactive simulation of fluids [9]. The SPH method represents a fluid as a set of discrete particles and models a resistance to density changes: when particles get too close to one another, a repulsive force separates them; when they get too far from each other, an attractive force brings them together. If a pair of particles is far enough apart, no forces act between them. SPH discretizes the Navier-Stokes equations and samples its solution at a finite number of such particles in space and time. While in the grid-based method the position of these sample points is fixed, in the SPH the particles are free to move around. This difference fundamentally changes the way the Navier-Stokes equations are solved and generally leads to much smaller complexity and ease of implementation, making SPH more suitable for interactive environments.

Cloth Simulation

Production physics: Cloth simulation models a cloth surface that can deform under the influence of external forces such as gravity or forced stretching, and internal forces such as the elastic response to tensile stress, shearing, and bending [2]. This application also models collisions of the piece of cloth with itself and other elements in the environment. The deformable cloth is modeled as a set of mass particles connected to form a triangle mesh. The mesh is endowed with a network of spring elements aligned with all triangle edges and altitudes, as well as between adjacent triangles. These springs model the cloth's resistance to various forms of deformation. Collision detection and resolution is a key part of this application. After the velocities and positions of the cloth particles are updated, collisions are detected. If the collisions cannot be resolved, the application undoes the updates from this iteration and re-executes it with a smaller time step.

Game Physics: Similar to production physics, game physics models a cloth object as a set of particles [7]. Each particle is subject to external forces, such as gravity, wind and drag, as well as various constraints. These constraints are used to maintain the overall shape of the object (spring constraints), and to prevent interpenetration with the environment (collision constraints). The particle's equation of motion resulting from applying the external forces is integrated using explicit Verlet integration. The above-mentioned constraints create a system of equations linking the particles' positions together. This system is solved at each simulation time step by relaxation, that is, by enforcing the constraints one after another for a fixed number of iterations. This method is less accurate but faster than the Conjugate Gradient solver [7] used in production physics, which enables the game cloth to simulate in real time. In addition, self-collisions are typically ignored.

Face Simulation

Production physics: Face simulation animates a model of a human face to provide an anatomically correct visualization of a person speaking or making facial expressions [12]. The application we examine assumes that inertia has a negligible effect on human faces in typical situations, and it therefore models facial motion as a sequence of steady states. Each state is defined by facial

muscle activation and the position of the cranium and jawbone. The face is modeled as a tetrahedral mesh, which is driven by the facial musculature and the motion of the jawbone. The application takes as input a time sequence of muscle activation values and kinematics parameters for the jaw motion. The finite element method is used to define the forces (elastic deformation resistance and active muscle contraction) that act on the face and determine its shape.

Rigid Body Simulation

Game Physics: Rigid body dynamics [3] simulates motion and interaction of non-deformable objects when forces and torques are present in the system. Rigid body dynamics is the most commonly used physical simulation in video games today. Examples of rigid bodies in games are vehicles, rag dolls, cranes, barrels, crates, and even whole buildings. The traditional approach solves a system of ordinary differential equations, which represent Newton's second law of motion, `F=ma`, where `m` is the mass of an object, `a` is its acceleration, and `F` is the applied force. The applied force determines the acceleration of the object, so velocity and position are obtained by integration of the above equation. The main computational challenge comes from the fact that rigid bodies' motion is constrained due to their interaction with the environment. For example, consider a destructive environment in a video game where 1000s of rigid objects explode, collapse, and collide, resulting in 100,000s of interactive contacts. To realistically simulate such a scene requires determination of collisions, calculation of collision contact points, and physically correct computation of the contact forces that result from these contacts. To accelerate collision detection relies on spatial partitioning data structures, such as grids or bounding volume hierarchies. To determine contact forces that result from collision contact, we model the contact as a linear complementary problem [1].

Rigid body simulation in games today assumes that rigid bodies cannot break. In general, this assumption is not true in production physics. Today's films use animation of elasticity and fracture. However, these techniques are too slow for interactive use.

PARALLELIZATION METHODOLOGY

The applications we study are all computationally demanding—on a 4-way Intel Xeon processor 3.0GHz system, with 16GB of DDR2-3200 and three levels of cache on each processor (16KB L1, 1MB L2, and 8MB L3), they take on average 188, 14, and 5 seconds to process a single frame for production fluid, face, and cloth simulations, respectively. Similarly, high-complexity scenes in game rigid body dynamics, fluid and cloth take on average 1, 0.4, and 0.1 seconds to process a single frame. While game physics performance may seem much better than production physics, one needs to perform at least 30 frames per second for real-time interactive experiences. Since they will all benefit from a large performance boost, we parallelize the applications, targeting a multi-core processor with tens of cores.

We took the conventional approach to parallelizing large code bases. We first profiled each application to determine the most expensive modules in a serial execution. After that, we prioritize the modules of each application and parallelize them in decreasing order of importance.

The applications were parallelized using the fork-join model in which the program consists of alternating serial and parallel sections. This model is attractive because it allows one to start with a serial program and selectively parallelize the most profitable portions of the program until satisfactory performance is achieved. We use a standard task queue technique [8], similar to Intel Thread Building Blocks (TBB) [6] and OpenMP [10], to parallelize all modules.

In the rest of this section, we discuss how the various modules were parallelized to scale to a large number of cores.

Parallelizing Loops

The majority of modules were parallelized via loop parallelization. These modules typically involve operations on arrays of elements, such as grid cells of a 3D grid (production fluid), an array of particles (game fluid), vertices of a triangle mesh (production cloth), and contact constraints (game rigid bodies).

In most of the cases, the iterations of the loops are independent of each other. For instance, computing the aggregate force on a vertex of the triangular mesh (production cloth) requires simply adding all the forces on that vertex. These loops are parallelized by partitioning the iterations of the loop among the cores. In a few instances, multiple iterations update the same piece of data. However, even in these instances, the final result is independent of the ordering of the iterations. These loops are also parallelized by splitting iterations among cores while using fine-grained locking to guard updates on the shared data.

Parallelizing Graph Operations

A few modules have more complex forms of parallelism and typically incur more parallelization overheads.

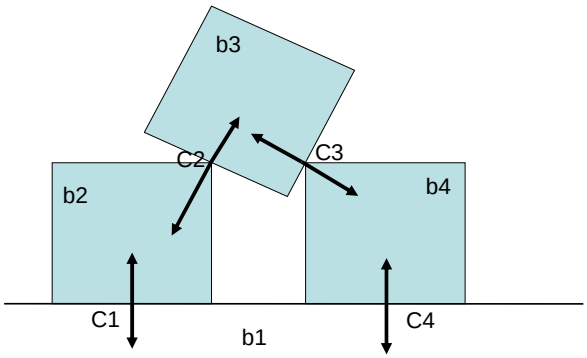


Figure 2: Scene configuration

Bodies → / Constraints ↓	b1	b2	b3	b4
C1	C1	C1		
C2		C2	C2	
C3			C3	C3
C4	C4			C4

Figure 3: Constraints between various objects

	b1	b2	b3	b4
Time 1	C1	C1	C3	C3
Time 2	C4	C2	C2	C4

Figure 4: Reordered constraints into two batches to expose parallel computation

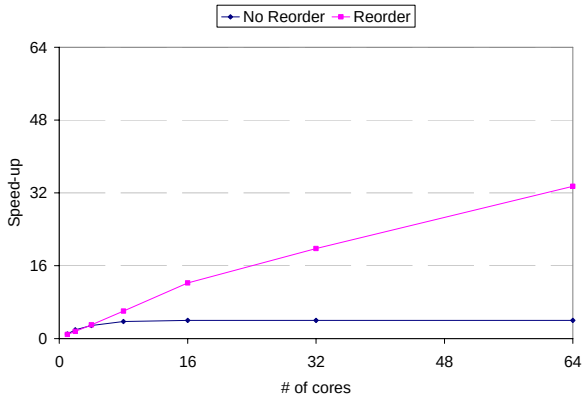


Figure 5: Relative speedups for the constraint solver with and without reordering of the constraints

An example of this is the broad-phase in collision detection. Collision detection requires checking every pair of objects for collisions. Since only a small fraction of the objects actually collide at any given instant, collision detection is performed in two phases: a *broad-phase* that performs quick checks to rule out a large fraction of the object pairs, and a *narrow-phase* that performs the exact (and more computationally expensive) check on the remaining pairs. A standard technique to accelerate the broad-phase is to build a bounding volume hierarchy (a tree) containing the objects. A leaf node of this tree consists of a single object. The computation starts at the root and traverses down. At each step, it checks pairs of nodes of the tree. If the bounding volumes at the two nodes do not overlap, then none of the objects in the first subtree can possibly collide with any object in the second subtree. Otherwise, more checks have to be performed using the children of the two subtrees. Each of these pairs of subtrees represents independent computation and can be performed in parallel. Consequently, in the broad-phase, each unit of parallel work can spawn off more parallel work.

Using Alternative Algorithms

Sometimes, the best serial algorithm has poor parallel scalability. In such cases, we often use an alternative algorithm whose serial version is not as efficient as the original, but whose parallel version scales much better. Sometimes, we use an additional phase to reorder data and expose more parallelism. In this section, we describe two specific examples in detail.

The first example is from rigid-body dynamics from game physics. During the execution of the physical simulation pipeline, the collision detection phase computes the pairs of colliding bodies, which are used as inputs to the constraint solving phase. The physics solver operates on these pairs and computes the separating contact forces, which keep the bodies from inter-penetrating each other. In Figure 2, we show one such case involving four bodies (three boxes and one ground plane), where the corresponding pairs of colliding bodies are listed in Figure 3. The resulting constraints C1, C2, C3, and C4 need to be resolved to update the body positions.

To parallelize this phase, we would ideally like to distribute the constraints amongst the available threads and resolve them in parallel. However, there is often an inherent dependency between consecutive constraints. In our example, constraints C1 and C2 both involve body b2 and thus cannot be resolved in parallel. These dependencies can force a significant serialization of the computation. However, we can reorder the constraints into different batches such that there are no conflicting constraints in each batch. That is, each batch will contain at most one constraint that refers to any given body.

Reordering algorithms traverse the constraints, maintaining an ordered list of partially filled batches. Each constraint is assigned to the earliest batch with no conflicting constraints. As a result, all constraints within a

batch can be processed in parallel, while the different batches have to be processed sequentially.

For example, we reorder the constraints in Figure 3 and obtain two batches, (C1, C3) and (C4, C2), as shown in Figure 4. Note that C1 and C3 from the first batch do not refer to any body more than once and can be resolved in parallel. A similar observation holds for C4 and C2. As a result C1 and C3 in the first batch are solved in parallel and the results are fed as part of the input to the second batch. The bottom curve in Figure 5 shows the speedup of the physics solver using the original order of the constraints relative to the serial version for up to 64 cores. The top curve shows the speedup using reordered constraints. We see that without reordering, the speedup is limited to 4× on 64 cores. However, reordering the constraints enables a speedup of 35×, including the overhead for reordering. This example highlights the case where some extra computation needs to be performed to expose the parallelism.

The second example of the need for alternative algorithms is from fluid simulation for production physics. Our fluid simulation application implicitly tracks the interface (boundary) between the air and the fluid. For each grid cell in the modeled space, it computes the distance to the interface. The most common technique to do this is the Fast Marching Method (FMM), which iteratively advances the wave front. For each iteration, it finds and updates the closest grid cell to the front that is not already on or behind the front (Figure 6). This is inherently serial. However, these distance values are required only for a narrow band around the interface. Thus, we parallelize FMM by dividing the grid into overlapping blocks, padded by the width of the narrow band, and working on each block independently (Figure 7). This works well for a small number of threads. However, the total overlap region becomes large quickly as the number of blocks increases. As a result, the application scales relatively poorly, achieving a scaling of around 21× on 64 threads.

We instead use an alternative scheme known as the Fast Sweeping Method (FSM) [5] (Figure 8). FSM traverses the grid cells in all eight possible combinations of the X, Y, and Z directions. Each "sweep" updates the distance of a cell from the distances computed for its neighbors in the previous sweeps. We obtain the correct distance for each cell after completing all eight sweeps. We parallelize FSM in a similar manner to FMM (i.e., with overlapping blocks). The serial version of FSM is around 30% slower than the serial version of FMM. However, FSM has more parallelism since the sweeps are independent. Thus, we achieve scaling of around 55× on 64 threads. In Figure 9, we compare the speedup of FSM relative to FMM. Up to 16 cores, FMM provides higher performance, but beyond 16 cores, FSM is better, giving about 2× the performance of FMM on 64 cores.

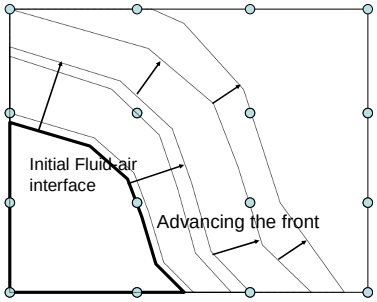


Figure 6: Fast Marching Method (FMM) advances the front to incrementally compute the signed distance of nodes from the interface

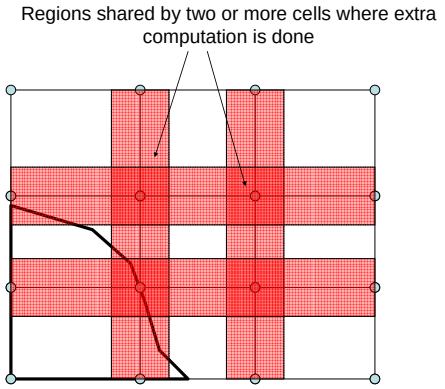


Figure 7: Parallelizing the algorithm by dividing the region into overlapping cells

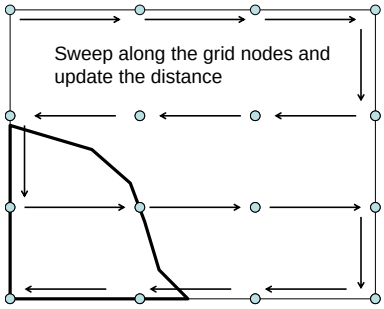


Figure 8: Fast Sweeping Method (FSM) traverses the grid nodes and incrementally updates the minimum distance to the interface

PARALLEL SCALABILITY RESULTS

Figure 10 shows the parallel scalability for our applications for up to 64 cores. Since no large-scale CMP is available for us to experiment with, we use cycle-accurate simulation to measure performance and characterize the parallelized workloads. Details of our simulator can be found in [5]. We assume a very high main memory bandwidth so that we do not artificially limit scalability. The speedups are obtained against the

one thread version of the parallelized code. On 64 cores, we achieve 30× to 56× speedup for production physics and 36× to 61× speedup for game physics.

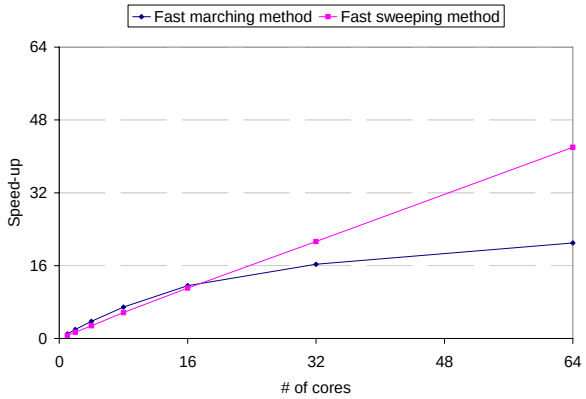


Figure 9: Speedup of FSM relative to FMM

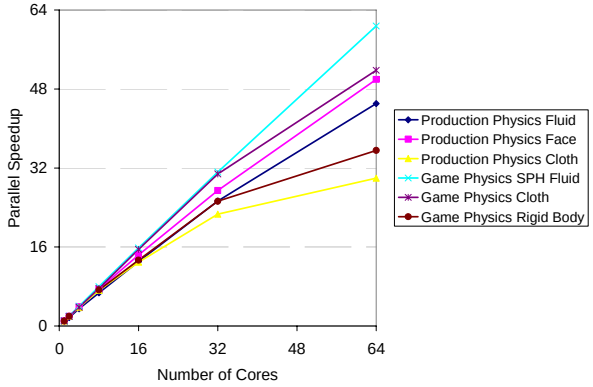


Figure 10: Parallel scalability of production physics and game physics

Next, we discuss important issues regarding scalability. Amdahl's law determines the theoretical maximum scalability. Load balancing and synchronization overheads impact how close we can be to the theoretical limit.

Serial Sections: Amdahl's law dictates that the parallel scalability is limited by the size of the serial sections. In most of the production and game physics modules, the serial section accounts for much less than 1% of execution time for one core. As a result, it does not significantly impact the parallel scalability in our study of up to 64 cores. However, as the number of cores increases, more aggressive parallelization will be needed to keep serial code from limiting parallel scalability.

Load Imbalance: The load imbalance is a function of the variability of task size as well as the number of tasks. The lower the variability, the fewer tasks are needed to obtain good load balance. Unfortunately, some modules exhibit high variability, which requires many tasks for good load balance, resulting in high parallelization overhead. Therefore, we should make a tradeoff between good load balance and low parallelization overhead. Under certain instances (e.g., Figure 7), we are forced to minimize the number of tasks to keep the amount of replication and redundant computation at an acceptable level. However, this comes at a cost of significant load imbalance that may limit parallel scaling.

Task Queuing: We implement a task queue to distribute parallel tasks across the cores. For some modules, the task queue overhead becomes a bottleneck for the scalability. In our implementation of task queues, all tasks are enqueued before we enter the parallel section. Therefore, if the number of tasks is large and/or the parallel section is small, the enqueue overhead becomes significant. Note that an alternative implementation of task queues might solve the problem, one of which is discussed in [8].

Locking: Grabbing and releasing locks incurs synchronization overhead. However, we observe that the locking overhead does not increase with the number of threads. Since there is little contention on the locks, locking does not significantly limit scalability. Nevertheless, accessing an uncontended lock still incurs parallelization overhead as it is extra work that is not required in a serial code.

In addition to the reasons listed above, parallel scaling is also affected by the memory behavior, which is covered in detail in the next section.

IMPLICATIONS FOR THE MEMORY SUBSYSTEM

Memory bandwidth requirements grow proportionally to the number of cores on a multi-core chip. Furthermore, as applications and workloads evolve, memory bandwidth requirements are expected to grow. Current server memory bandwidth projections are mostly based on traditional benchmarks such as TPC-C, SPECjAppServer (SJAS), and SPECjbb (SJBB). Unfortunately, these benchmarks do not accurately reflect future important workloads such as our physical simulation applications.

Figure 11 shows the projected external memory bandwidth requirements for five different sizes of last-level cache (the other caches are assumed to be small and inclusive). The projection is based on running the workloads at 64 giga-instructions-per-second (GIPS). We analyze the bandwidth requirements for all important modules and compare them to TPC-C, SJAS, and SJBB. For each cache size, the modules are sorted according to their bandwidth requirements. The bandwidth requirements for the traditional benchmarks are highlighted for comparison.

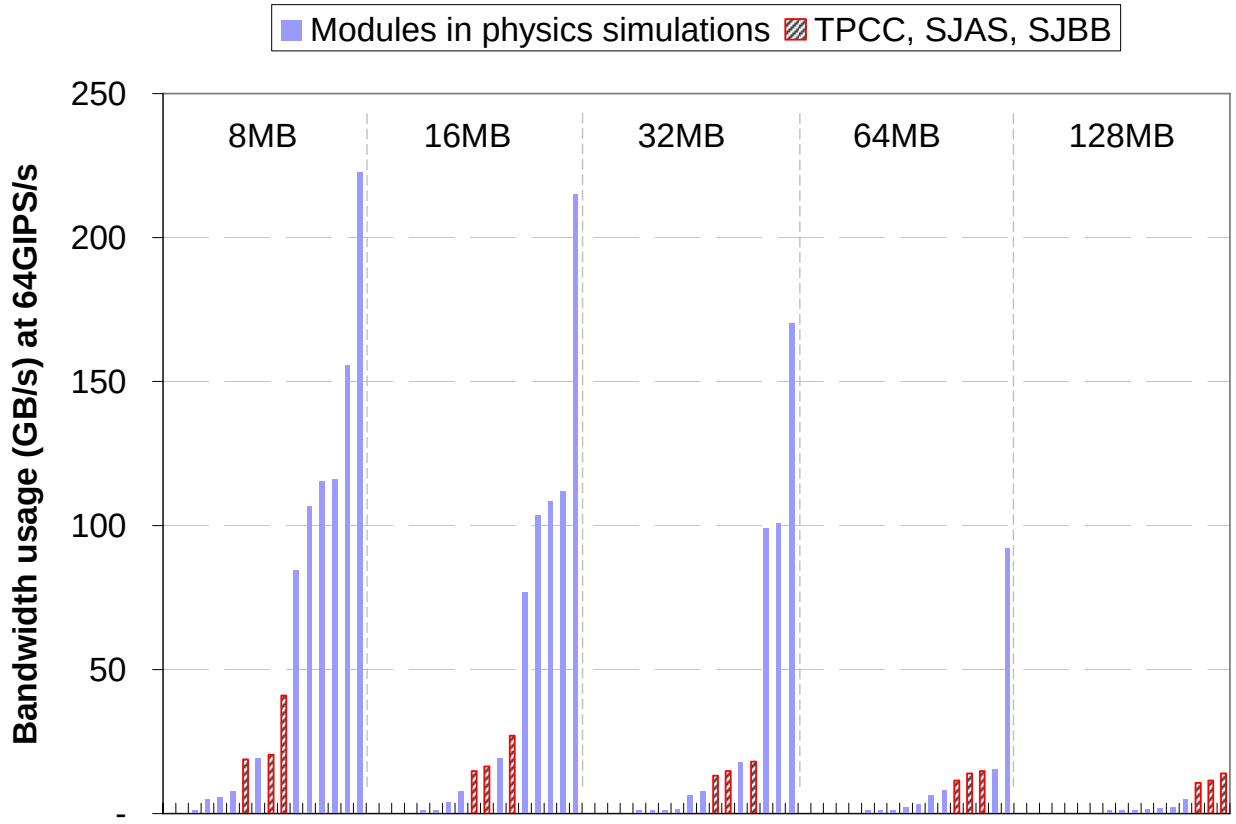


Figure 11: Projection of external memory bandwidth requirements (GB/s) for a given last-level on-die cache size

The results show the following behaviors:

(1) If we have less than 128MB of last-level cache, modules in physical simulation have a wide range of bandwidth requirements, ranging from a few gigabytes per second to over 200GB/s. The bandwidth usage of traditional benchmarks, on the other hand, is much lower than that (maximum of 40GB/s, even if we have only 8MB of cache).

(2) To put the results into context, we compare the requirements to projected available bandwidth in 2010. Memory bandwidth typically grows at 30% per year, so we expect the available bandwidth to be about 48GB/s in 2010. Workloads with bandwidth requirements greater than this will suffer performance-wise. Some of our modules have bandwidth requirements that greatly exceed 48GB/s unless the last-level cache is at least 64MB.

(3) The average bandwidth usage for each of the applications is significantly lower than the peak bandwidth usage. This is because each application is made up of modules with different bandwidth requirements. The scalability of the module with the highest bandwidth requirement often limits the scalability of the entire application.

(4) Our physical simulation modules benefit significantly more than traditional benchmarks do from a large last-level cache. When an application's entire working set fits into cache, the external memory bandwidth usage becomes minimal. For our applications, this happens when the cache is 128MB.

(5) One of our most memory-intensive modules is the incomplete Cholesky Preconditioned Conjugate Gradient (PCG) method from production fluid simulation. PCG is used to solve a system of equations arising from the discretization of the Poisson Equation.[1] It consists of a number of

[1] PCG is one of the most popular approaches for solving large symmetric positive-definite systems of equations because it is more robust than direct solvers and converges fast. As such, PCG is of great importance beyond the study of this application.

operations performed sequentially on a set of two matrices and a number of vectors. The solver iterates tens of times until the solution converges. During each iteration, both matrices (which occupy about 40MB each) are streamed over. Thus, we see a huge bandwidth requirement when the last-level cache cannot hold the matrices. When the last-level cache is big enough to hold both matrices (and all the vectors), the bandwidth requirement is greatly reduced.

CONCLUSION

We consider two broad categories of physical simulation applications: production physics and game physics. Production physics is used by movie studios for creating special effects that may take many minutes to process a single frame. In contrast, game physics is used by the gaming industry and has a more stringent real-time requirement of about 30-60 frames per second. The difference in execution time requirements affects the choice and design of algorithms for the two categories of physical simulation.

We have parallelized applications in both categories and achieve parallel scalability of 30-60× on a cycle-accurate simulator of a multi-core chip with 64 cores. Many modules of these applications require extensive effort to achieve good performance scaling. In some cases, the best serial algorithms have poor parallel scalability. For these, we use alternative algorithms which are slower on one core, but have more parallelism. In other cases, we modify the algorithm to expose more parallelism. The overhead of exposing the parallelism is often small compared to the benefits of improved scaling.

While our applications scale well, some modules are far from the theoretical maximum scaling. This is primarily due to overheads in the task queues and to imperfect load balancing.

Some modules also have significant overheads from locking, but these overheads do not grow with the number of cores (i.e., the locks have low contention), and therefore do not impact scalability. However, the cost of locking still has a significant impact on the overall performance of the parallelized application.

We find that future physics workloads will require large last-level caches (i.e., 128MB) or main memory bandwidths in excess of 100GB/s. This is due to the applications' use of streaming access patterns combined with large data sets (e.g., tens of thousands of objects for game physics and hundreds of thousands to a few million objects for production physics).

We also find that physical simulation applications have very different memory characteristics than traditional benchmarks such as TPC-C, SPECjAppServer, and SPECjbb. These traditional benchmarks do not get a big boost from a large last-level cache since their working sets are extremely large. However, physical simulation applications benefit greatly from a 128MB cache since it can fit the whole working set of all application modules.

ACKNOWLEDGMENTS

We thank Radek Grzeszczuk, Matthew J. Holliman, Richard Lee, Andrew P. Selle, and Jason Sewall for their contributions to the project. We also thank Pradeep Dubey who encouraged us to look into this problem.

REFERENCES

[1] D. Baraff, "Physically Based Modeling: Principals and Practice," *Online Course Notes, SIGGRAPH*, 1997.

[2] R. Bridson, S. Marino, and R. Fedkiw, "Simulation of Clothing with Folds and Wrinkles," in *Proceedings of the Eurographics Symposium on Computer Animation*, 2003.

[3] D. H. Eberly, *Game Physics*, Morgan Kaufmann/Elsevier, San Francisco, 2003.

[4] D. P. Enright, S. R. Marschner, and R. P. Fedkiw, "Animation and Rendering of Complex Water Surfaces," *ACM Transactions on Graphics*, 21(3):736–744, July 2002.

[5] C. J. Hughes, R. Grzeszczuk, E. Sifakis, D. Kim, S. Kumar, A. P. Selle, J. Chhugani, M. Holliman, and Y.-K. Chen, "Physical Simulation for Animation and Visual Effects: Parallelization and Characterization for Chip Multiprocessors," in *Proceedings of the 34th International Symposium on Computer Architecture*, June 2007.

[6] *Intel® Thread Building Blocks Reference*, 2006, Version 1.3.

[7] T. Jacobsen, "Advanced Character Physics," *Game Developers Conference*, 2001.

[8] S. Kumar, C. J. Hughes, A. Nguyen, "Carbon: Architectural Support for Fine-Grained Parallelism on Chip Multiprocessors," in *Proceedings of the 34th International Symposium on Computer Architecture*, June 2007.

[9] M. Muller, D. Charypar, and Markus Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the Eurographics Symposium on Computer Animation*, 2003.

[10] *OpenMP Application Program Interface*, May 2005, Version 2.5.

[11] *PhysBAM physical simulation package*, at http://graphics.stanford.edu/~fedkiw*

[12] E. Sifakis, A. Selle, A. Robinson-Mosher, and R. Fedkiw, "Simulating Speech with a Physics-Based Facial Muscle Model," in *Proceedings of the Eurographics Symposium on Computer Animation*, 2006.

[13] R. Smith, "*Open Dynamics Engine*, at http://www.ode.org*

AUTHORS' BIOGRAPHIES

Yen-Kuang Chen is a Principal Engineer in the Corporate Technology Group. His research interests include developing innovative multimedia applications, studying the performance bottleneck in current architectures, and designing next-generation microprocessors/platforms. He is one of the key contributors to Supplemental Streaming SIMD Extension 3 in the Intel® Core™2 processor family. He received his Ph.D. degree from Princeton University. His e-mail is yen-kuang.chen at intel.com.

Jatin Chhugani is a Staff Researcher in the Corporate Technology Group. His research interests include developing algorithms for interactive computer graphics, parallel architectures, and image processing. He received his Ph.D. degree from The Johns Hopkins University, Baltimore, MD. His e-mail is jatin.chhugani at intel.com.

Christopher J. Hughes is a Staff Researcher in the Corporate Technology Group. His research interests are emerging workloads and computer architectures, with a current focus on parallel architectures and memory hierarchies. He received his Ph.D. degree from the University of Illinois at Urbana-Champaign. His e-mail is christopher.j.hughes at intel.com.

Daehyun Kim is a Senior Research Scientist in the Corporate Technology Group. His research interests include parallel computer architecture, intelligent memory systems, and emerging workloads. He received his Ph.D. degree from Cornell University. His e-mail is daehyun.kim at intel.com.

Sanjeev Kumar is a Staff Researcher in the Corporate Technology Group. His research interests are parallel architectures, software, and workloads especially in the context of chip-multiprocessors. He received his Ph.D. degree from Princeton University. His e-mail is sanjeev.kumar at intel.com.

Victor Lee is a Senior Staff Research Scientist in the Corporate Technology Group. His research interests are computer architecture and emerging workloads. He is currently involved in defining next-generation chip-multiprocessor architecture. He received his S.M. degree from the Massachusetts Institute of Technology. His e-mail is victor.w.lee at intel.com.

Albert Lin is a graduate intern in the Corporate Technology Group. His work is primarily in memory systems for future processors with many cores. He received his B.S. and M.Eng. degrees in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology. While at MIT, he was a recipient of the Siebel Scholar Fellowship and a student member of the American Academy of Achievement. He is currently an Electrical Engineering doctoral candidate at Stanford University. His e-mail is albert.c.lin at intel.com.

Anthony D. Nguyen is a Senior Research Scientist in the Corporate Technology Group. His research interests include developing emerging applications for architecture research and designing the next-generation chip-multiprocessor systems. He received his Ph.D. degree from the University of Illinois, Urbana-Champaign. His e-mail is anthony.d.nguyen at intel.com.

Eftychios Sifakis is a visiting researcher in the Corporate Technology Group. He received B.Sc. degrees in Computer Science and Mathematics from the University of Crete, Greece in 2000 and 2002, respectively, and he received his Ph.D. degree in Computer Science from Stanford University in 2007. His research focuses on simulation and analysis of human body and face motion and simulation algorithms for deformable solids. He has been working with Intel since 2005 on the mapping of physics-based simulation on chip-multiprocessors. His e-mail is eftychios.d.sifakis at intel.com.

Mikhail Smelyanskiy is a Senior Research Scientist in the Corporate Technology Group. His research focus is on building and analyzing parallel emerging workloads to drive the design of the next-generation parallel architectures. He received his Ph.D. degree from the University of Michigan, Ann Arbor. His e-mail address is mikhail.smelyanskiy at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

For further information visit:

developer.intel.com/technology/itj/index.htm